Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

2001-06

Methods for determining object correspondence during system integration

Pugh, Randolph G.

http://hdl.handle.net/10945/10933

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

METHODS FOR DETERMINING OBJECT CORRESPONDENCE DURING SYSTEM INTEGRATION

by

Randolph G. Pugh

June 2001

Thesis Advisor: Valdis Berzins
Second Reader: Paul Young

Approved for public release; distribution is unlimited

20010831 097

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Methods for Determining Object Correspondence during System Integration			**5. FUNDING NUMBERS**
6. AUTHOR(S) Randolph G. Pugh, Captain, United States Marine Corps			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Object correlation is a semantic comparison of exported entities from one system to imported entities of another. Current research in search algorithms and artificial intelligence methods for pattern matching can aid integrators in finding these matches. This thesis proposes a two-stage correlation process for resolving various kinds of heterogeneity found in legacy DoD systems to facilitate interoperability. A prototype built using these methods is explained, results compared to current correlation methods, and recommendations made for further improvements.

The end of the Cold War and the Defense Reorganization Act of 1986, began a new era of unprecedented cooperation among the U.S. military services and our allies. Increasingly dynamic missions have required warfighters to share information quickly and seamlessly while a decreasing defense budget has left few resources to build the infrastructure needed to implement this information exchange in legacy heterogeneous data systems. One possible solution to achieving interoperability of information systems is Young's Federated Interoperability Model. This model allows system designers to advertise the kinds of information they produce and consume and then automatically provides translation services. Before data and services can be shared, however, integrators must resolve exactly what kinds of data they are providing so that other systems in the network can decide if that data is appropriate for their use. That is the purpose of the proposed correlation algorithm.

14. SUBJECT TERMS Command, Control, and Communications, Computing and Software			**15. NUMBER OF PAGES** 90
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

METHODS FOR DETERMINING OBJECT CORRESPONDENCE DURING SYSTEM INTEGRATION

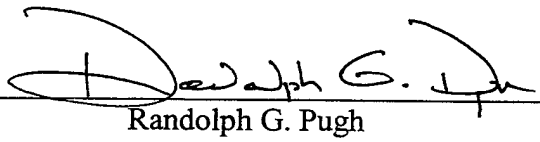
Randolph G. Pugh
Captain, United States Marine Corps
B.S., United States Naval Academy, 1994

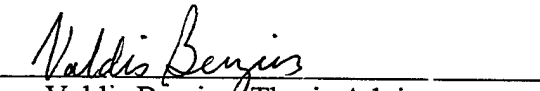
Submitted in partial fulfillment of the
requirements for the degree of

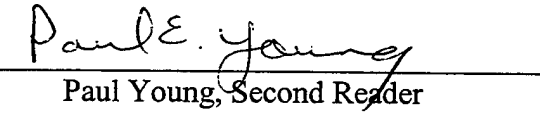
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2001

Author: 
Randolph G. Pugh

Approved by: 
Valdis Berzins, Thesis Advisor


Paul Young, Second Reader

Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Object correlation is a semantic comparison of exported entities from one system to imported entities of another. Current research in search algorithms and artificial intelligence methods for pattern matching can aid integrators in finding these matches. This thesis proposes a two-stage correlation process for resolving various kinds of heterogeneity found in legacy DoD systems to facilitate interoperability. A prototype built using these methods is explained, results compared to current correlation methods, and recommendations made for further improvements.

The end of the Cold War and the Defense Reorganization Act of 1986 began a new era of unprecedented cooperation among the U.S. military services and our allies. Increasingly dynamic missions have required warfighters to share information quickly and seamlessly while a decreasing defense budget has left few resources to build the infrastructure needed to implement this information exchange in legacy heterogeneous data systems. One possible solution to achieving interoperability of information systems is Young's Federated Interoperability Model. This model allows system designers to advertise the kinds of information they produce and consume and then automatically provides translation services. Before data and services can be shared, however, integrators must resolve exactly what kinds of data they are providing so that other systems in the network can decide if that data is appropriate for their use. That is the purpose of the proposed correlation algorithm.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. PURPOSE 1
B. SETTING THE STAGE: THE COLD WAR 2
C. DOING MORE WITH LESS 4
D. THE HOLY GRAIL OF FULL AND SEAMLESS INTEGRATION 5
E. BENEFITS OF SYSTEM INTEGRATION 6

II. ENABLING INTEROPERABILITY 9
A. BACKGROUND 9
B. WHAT DOES HETEROGENEOUS MEAN? 9
C. WAYS TO INTEGRATE SYSTEMS 13
D. SOFTWARE INTEGRATION OPTIONS 14
1. Reengineering Legacy Code 16
2. Software Wrappers 18
3. Lingua Franca Automata 20
E. FEDERATION INTEROPERABILITY OBJECT MODEL 21
1. Building the FIOM 24
2. Using the FIOM 28
F. SUMMARY 29

III. CORRELATION METHODOLOGY 31
A. BACKGROUND 31
B. WHAT CONSTITUTES ENTITY CORRESPONDENCE? 31
C. CORRELATION EFFECTIVENESS 34
D. CONVENTIONAL SEARCH METHODS 35
1. Browsers 35
2. Keyword Search 36
3. Faceted Classification 37
4. Semantic Integration (SEMINT) Tool 37
D. SUMMARY 40

IV. FEDERATION INTEROPERABILITY CLASS CORRESPONDENCE 41
A. ASSUMPTIONS 41
1. Existence of Detailed XML Schemas 42
2. Domain Expert Participation 43
B. OVERVIEW OF CORRELATION PROCESS 44
C. NORMALIZING AND INDEXING THE SEARCH SPACE 45
1. Gathering Syntactic Information 46
2. Determining Entity Semantics using SEMINT 47
a. Building Representative Vectors 48
b. Clustering Attributes 50
c. Training the Neural Network 51
D. PERFORMING OBJECT CORRELATION 53
E. SUMMARY 55

V. CONCLUSIONS 57
A. EFFECTIVENESS 57
B. RECOMMENDATIONS FOR FUTURE RESEARCH 59
C. CONCLUSIONS 60
APPENDIX A: DETAILED XML SCHEMA (PARTIAL) 61
APPENDIX B: SAMPLE KEYWORD LIST 63
APPENDIX C: JAVA CODE - KEYWORD GENERATOR 65
LIST OF REFERENCES 69
INITIAL DISTRIBUTION LIST 71

LIST OF FIGURES

Figure 1. Typical External Interface of a Software Module 15
Figure 2. Interaction of Software Modules Through Interfaces 15
Figure 3. Reengineering Legacy Code to Match Interfaces 16
Figure 4. Use of Software Wrappers to Match Interfaces 18
Figure 5. Pair-wise Software Wrapper Construction 19
Figure 6. Benefits of Common Representation in Wrapper Construction 20
Figure 7. Modification of the Contemporary Object Class Diagram 22
Figure 8. Location of Entities in Legacy Formatted Messages 25
Figure 9. XML Representation of XPOS Message 26
Figure 10. Composition of FICs 27
Figure 11. Message Passing Using the FIOM From Ref. [Young01] 29
Figure 12. Correlation Tool Showing Results of Syntax Match 45
Figure 13. Discriminator Vector for date_time_group Attribute 49
Figure 14. Self-Organizing Map 50
Figure 15. Typical Back-Propogating Neural Network 52
Figure 16. Matrix of Attribute Comparisons Using SEMINT Tool 54
Figure 17. Calculating the Entity Average SEMINT Score. 54

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1. Comparison of Location Representations.....10
Table 2. List of XML Schema Fields Used for Keyword Determination.....47
Table 3. XML Schema Fields Used for Discriminators. 49

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I owe thanks to many people, not just those who actively helped me with this thesis, but also to the countless others who constantly reminded me that it was worth doing. The following list is necessary but not sufficient...

- Ann, my wife, for holding down a fort filled with three wild Indians while I was at school. I'll always remember that the effort required to earn the title "MS" pales in comparison to the effort required to earn the title "Mommy." Also my children Nicholas, Savannah, and Morgan who helped me keep the important things (scooter riding, ice cream, and story time) in perspective.
- Captain Young. Whose patience, vision, and guidance pushed me to finish a thesis while and still find time to pass along his thoughts on the complexities of interoperability. Hopefully -- one day -- he can stop feeding the lion.
- The other students in the CS-01 section, especially Maj. Brent Christie, my partner in crime. Although I'm proud to graduate, I'm more proud of having known you and learned from you.

"We few, we happy few, we band of brothers;
For he who sheds his blood with me this day shall be my brother."
-- *Henry V*, William Shakespeare

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PURPOSE

The purpose of this thesis is to develop a fast and flexible data correlation algorithm that will facilitate legacy Department of Defense (DoD) system information exchange. The algorithm enables a larger interoperability model and its associated integration software tool and will assist system integrators in identifying where disparate systems share real-world concepts.

In this chapter, some of the reasons why the DoD has so many stove-piped legacy systems are explained, ways in which the DoD could benefit from a computer assisted integration mechanism are explored, and how such a system could fundamentally transform the information quality and quantity for all warfighters is proposed.

Chapter II details Young's Object Oriented Model for Interoperability (OOMI). The challenges of building a flexible and extensible system that allows unrestricted information sharing are covered and the benefits of this particular solution are promoted.

Chapter III contains a review of current techniques used by integrators for determining object correlation between heterogeneous systems. The advantages and disadvantages of various state-of-the-art search methodologies are detailed and commercially available tools are mentioned.

Chapter IV details the theory and implementation of the improved semantic correlator used in Young's Federation Interoperability Object Model Integrated

Development Environment (F-IDE). Design considerations are explained and the benefits of this solution are demonstrated.

Chapter V provides concluding remarks about the current system state. Pitfalls and potentials for further research are mentioned.

B. SETTING THE STAGE: THE COLD WAR

For over two hundred and twenty-five years, the United States government has recognized military strength as a necessity for self-protection and as an effective instrument of foreign policy. During this time, policymakers have recognized America's technical superiority with military might. Examples abound: Admiral Perry's steam-powered ships anchoring in Tokyo Harbor to open trade with Japan, the Great White Fleet's cruise around the world to ensure Europe's respect for the Manifest Destiny, and the Enola Gay dropping the Hiroshima bomb to end a global war and thrust America into a dominant position in world politics. Cutting-edge technology, it was believed, equated to national strength.

In the belief that the next war would be won through technology, America spent large portions of her national treasure to further the state of military science. A fundamental shift in the nature of warfare in the early Twentieth Century to one of "total war" meant that this effort increasingly involved civilians and civilian manufacturers. These institutions quickly became stakeholders in future conflicts since they relied on strong national security for their very existence. By as early as the end of World War II, the military's technical interests significantly overlapped that of the civilian sector.

An unexpected benefit of this relationship soon became apparent. The military's research, originally exclusively for national defense, increasingly benefited the nation as a whole. Transfers of technology from military applications to civilian had a profound effect on the overall national paradigm. Technologies such as nuclear energy, satellite technology, advanced communications theory, Radio Direction and Ranging (RADAR) equipment, and the Global Positioning System (GPS) soon found multiple and diverse uses and changed Americans' daily lives. American businesses also found that the technology transfer worked in the opposite direction as well and modification or direct reuse of civilian products improved military capabilities. Soon distinctions between military and civilian technology were almost insignificant and Commercial Off-The Shelf (COTS) procurement became "best-practice" for acquisition professionals.

By the second half of the Twentieth Century, the United States and the Soviet Union (USSR) had divided the world into two immense spheres of influence. Conventional wisdom was that since the two powers had such different ideologies, defeat for the United States would mean complete cessation of its way of life. Such high stakes necessitated that the DoD continue to push the technological barriers as far and as fast as possible. Our enemy, the USSR, had immense physical size, population, and manufacturing capability. The American military would not have conventional force parity. To make up the difference in numbers, the DOD looked to superior technology as a combat multiplier. In a search for the "silver bullet," new technologies or academic theories, which might have military application, were implemented in a working system as quickly as they appeared. Whether to listen to, silence, take pictures of, or kill our enemies, speed was of the essence.

The fall of the Berlin Wall in 1989 ended the bi-polar political world and the Cold War. It also ended the arms and technology race between the USSR and the United States. Unfortunately, this war did not have a clear victor. In the 1990's, without the oversight of the United States or the USSR, rogue nations, terrorist groups and non-state actors proved that they could and would act independently. Over the next decade, these groups would prove to be a serious threat requiring new techniques and technologies, equally as sophisticated as those used against the USSR, to understand and counter them. Interoperability at this point was almost non-existent as all the services had fielded expeditiously engineered software and hardware so that troops would have the latest technology on the front lines as quickly as possible.

C. DOING MORE WITH LESS

In the late 1980s, because of a proliferation of autonomously developed, expensive, yet non-interoperable systems, the Defense Reorganization Act of 1986, also known as the Goldwater-Nichols Act, dictated that the services start moving toward increased "jointness" [GN86]. While the individual services would remain, removing redundancies among them would improve efficiency. To date the services have done a commendable job of resolving their cultural differences and building on each other's strengths in important areas. The massive success of Operation Desert Storm was largely due to the compromises made by every service in regards to equipment, tactics, techniques, and procedures. Unfortunately, system interoperability has not seen the same advances.

Success in Desert Storm sent a different message to the American people. The taxpayers, mistakenly believing the end of the Cold War meant victory for the United

States, decided it was finally time to scale back the bloated military they had supported for over four decades. Budget cuts and manpower caps for all the military services reduced both forces and programs. Americans, however, were **not** ready to give up America's new role of sole super-power. Following Desert Storm, America was left looking for ways to maintain technical superiority in a climate of decreasing resources. At a time when the benefits of interoperability were clearly understood, limited funds necessitated retrofitting legacy systems instead of building interoperable systems from the ground up.

D. THE HOLY GRAIL OF FULL AND SEAMLESS INTEGRATION

Throughout the DoD, thousands of legacy systems, currently operating in a stand-alone configuration or within a closed system, continue to perform various important functions. There are many reasons why so many non-integrated systems exist. These include:

- The original intended system use was so restricted that there was no need to integrate;
- The original system was considered such a technological advantage over our enemies that it was classified at a security level above the majority of other existing systems;
- In the rush to build a better system or fill a technological gap, consideration for future integration was overlooked;
- The original system designers never imagined that their system's lifecycle would extend as long as it has;
- The system was designed by contractors who did not, or would not, communicate.

For the reasons mentioned above, many of these systems run on proprietary hardware and almost all of them execute proprietary software. Clearly, if we could get all these "boxes" to communicate, sharing their information and capabilities would exponentially increase America's warfighting capability. The synergistic effect found in joint operations and realized in Desert Storm, would finally apply to systems as well. The United States intelligence community already advertises a "system of systems" (i.e. independent systems doing specific jobs sharing their information as part of a larger system) as an essential aspect of United States' national intelligence strategy.

E. BENEFITS OF SYSTEM INTEGRATION

The implications of legacy system integration are profound and far-reaching. For example, consider a simple navigation system that currently relies on a proprietary inertial guidance system for self-location. If a way existed to "plug" a commercial GPS feed into the navigation software, it would tremendously increase the accuracy and reliability of the older system. In addition, the old system, requiring proprietary parts and special training, would no longer be needed. Precious maintenance dollars and man-hours could be invested elsewhere or even given back to the taxpayer.

Cheaply upgrading old equipment is only one benefit. The other is information sharing beyond what has ever been possible in the past. DoD systems exist at varying levels of sophistication, even when they perform the same mission. Obviously, every military commander would like to have the latest Joint Surveillance and Target Attack Radar System (J-STARS) aircraft flying in direct support of their unit. Unfortunately, this multi-million dollar collection system requires a specialized downlink station to receive and analyze data. Due to limited numbers, each ground station can only serve

one high-level headquarters. Integration of this system would allow the information gathered from the J-STARS to be passed to "light weight" intelligence analysis systems. Lower level units would then have an unsurpassed intelligence source with no additional cost or effort.

The United States can no longer afford to build single-purpose, specialized, stove-piped systems. The immense expense of building sophisticated systems requires that producers' information be made available to every consumer that might need it. In addition to the thousands of legacy systems that are limited by non-integration, new systems often rely on proprietary COTS software and hardware that was not designed with interoperability in mind. If these systems had a free and flexible means for communication, America's warfighters would be more efficient, informed, and, ultimately, capable.

THIS PAGE INTENTIONALLY LEFT BLANK

II. ENABLING INTEROPERABILITY

A. BACKGROUND

The benefits of legacy system integration are becoming clearly understood in the DoD. Since Goldwater-Nichols, acquisition professionals are required by law to pay close attention to how well a system will integrate with other systems [GN86]. Backward compatibility must be maintained and forward compatibility must be planned for. These same issues apply to legacy system integrators but without the benefit of being able to design for interoperability from the ground up. Before proposing a satisfactory method for integration, one must understand the various reasons why systems might not interoperate.

B. WHAT DOES HETEROGENEOUS MEAN?

[Web97] defines *heterogeneous* as "consisting of or composed of dissimilar elements or ingredients; not having a uniform quality throughout." This definition, while adequate for human understanding, does not sufficiently define the scope of the issues facing legacy system integrators.

The existence of independently designed, physically separated systems almost always means multiple representations of the same entity exist. For example, two systems that both keep track of locations may represent the data differently depending on how the data will be used and by whom. Table 1 illustrates this phenomenon in the Advanced Field Artillery Tactical Data System (AFATDS), used for assigning artillery fire missions, and the Air Force's *FalconView,* used by aviators for mission planning. In

the example, the "location" field represents the actual place on the ground where you can find the Salinas airport.

System	Use	Location	Representation
AFATDS	Artillery fire mission targeting	32RG13570378	Military Grid Reference System (UTM)
FalconView	Air Mission Planning	27.25N/128.532W	Latitude/Longitude (Lat/Long)

Table 1. Comparison of Location Representations

Although both systems agree on *what* they are representing and often even the *level of detail* which they will represent it, each system chooses a different way to store, display, and process the data. Until these differences are resolved, systems cannot share their information. This is only one small sample of the challenges facing legacy system integrators. DoD automated systems demonstrate different kinds of heterogeneity at multiple levels. In [Wie93] Wiederhold proposed that heterogeneity could exist in hardware, organizational models, representation, scope, levels of abstraction, meaning, and temporal validity. While all of these issues may face a system integrator, our research only attempts to address a subset of the possible kinds of heterogeneity.

Heterogeneity of hardware and operating systems at the physical layer may mean that data is represented in the computer differently depending on system implementation. This is the most fundamental task since there must exist a way to pass messages between systems before we can start to address heterogeneity in the information contained in the message. This type of heterogeneity is not addressed in this thesis but is noted as a potential area for further research.

Heterogeneity of organizational models is still a problem for database integrators trying to merge object-oriented and relational databases. To limit the scope of this thesis, the systems we intend to target for integration pass text-based messages. Currently, these text messages are being converted to include eXtensible Markup Language (XML) tags. Although the entity tag names may differ between systems, XML provides a homogeneous syntax for every message, regardless of producer. This kind of heterogeneity is assumed resolved and will not be addressed in this thesis.

Heterogeneity in representation poses significant challenges for legacy system integrators. All systems expect data to be in a certain format. Even the simple example of AFATDS and *FalconView* illustrates how complicated this conversion might be. Bandwidth saving practices such as enumerating commonly used words or phrases, and using cryptic abbreviations or numbers, complicates the resolution of similar objects and translations between the two systems. In chapter IV, we propose ways to mitigate this kind of heterogeneity.

Heterogeneity of scope refers to the differing amount and type of data each individual system feels it needs to capture for its level of abstraction. Because legacy systems were designed for specific functionality, different systems often represent the same real-world object at different levels of detail. When these systems are integrated, the possibility exists that a consumer system may be presented with an incomplete representation. This thesis addresses this kind of heterogeneity.

Heterogeneity of levels of abstraction will occur if different views *within a single system* will cause the loss of atomic data about objects. Since this thesis only deals with external messages, this type of heterogeneity does not apply.

Heterogeneity of meaning is possibly the greatest challenge for integrators and also the most important to resolve. Any computer assistance to facilitating interoperability must identify where systems can share information about real-world entities and then facilitate that transfer of information. To do this requires an understanding of the semantics of the entity each system represents. The main purpose of this thesis is to automate finding semantically similar entities in non-integrated systems. This process is covered in chapter IV.

Heterogeneity of temporal validity refers to the assumed fact that the entities of interest will constantly change. This may not be a problem for messages themselves since the information is considered constant once it leaves the producing system. For interoperability, however, the data contained within the message may only be useful, or even relevant, for a defined period of time. In some stand-alone systems the temporal validity of data is treated **implicitly**. When this system is integrated into a larger federation of systems, however, temporal restrictions must be stated **explicitly**. This problem is only partially addressed by Young's Object Oriented Model for Interoperability covered in chapter III.

Regardless of the kind of heterogeneity in entity representation, humans are able to correlate various entities because of an innate ability to abstract a problem to a level where heterogeneity issues resolve themselves. Whether consciously or unconsciously,

system integrators understand that these abstractions carry with them risks. Previous attempts at integration used the human capacity for abstraction to resolve problems of heterogeneity. Domain experts examined the systems they wanted to interoperate, decided what real world entities were shared, and then decided on the best way to transfer the information. This process, while often successful, was prohibitively slow and complicated. In large database integration efforts, correlating entities by hand could take up to four hours per attribute[Clif99]. To provide a useful tool for integration, we must somehow automate this process of abstraction and risk calculation without the benefit of a domain expert negotiating each and every transaction.

C. WAYS TO INTEGRATE SYSTEMS

There are three ways of integrating systems; change one or both system's hardware, change one or both system's software, and through organizational shift. Currently, the prevalent method is organizational shift. System operators have devised work-arounds and *ad hoc* methods to get useful information from one non-integrated system to another. Transcribing information from a producer system to a piece of paper and then inputting it into another system is common practice. For most situations this solution is unacceptable since it introduces errors in the data, creates multiple instances of the same object, and takes an excessive amount of time. Automation of the process is clearly needed.

For some systems, changing hardware is the cheapest and most effective way to resolve system incompatibilities and for some types of heterogeneity, it may be the only method. This is a "heavyweight" method, however. This solution requires extensive engineering talent to completely understand the target systems and then design, build, and

manufacture a device to facilitate integration. A knowledgeable system engineer then has to install the device locally on every system in the federation. Training and maintenance may be involved. Once the device is in place, any additional changes will require repeating the entire process for every system.

For most systems the inherent flexibility of software makes it the prime candidate for carrying the burden of interoperability. This flexibility can also present problems, though. Building software is largely an intellectual exercise carried out by imperfect humans. Improvements in software engineering have made the process of writing applications more of an engineering exercise, but the majority of the systems in the DoD were built years, if not decades, ago using rigid procedural languages and immature techniques. The challenges of using software to achieve system interoperability, while less than hardware, are still significant.

D. SOFTWARE INTEGRATION OPTIONS

Integrating systems through the use of software is an appropriate way to address the different types of heterogeneity among systems. When this method is chosen, the system integrator usually concentrates on one or both of the systems' external interfaces. External interfaces are where the system interacts with other systems or, for internal modules, how the modules interact with each other. Interfaces are communication conduits. As an example, Figure 1 shows the external interfaces for a location calculation module. The module makes available to other internal modules or other systems, the time, longitude, and latitude for a contact.

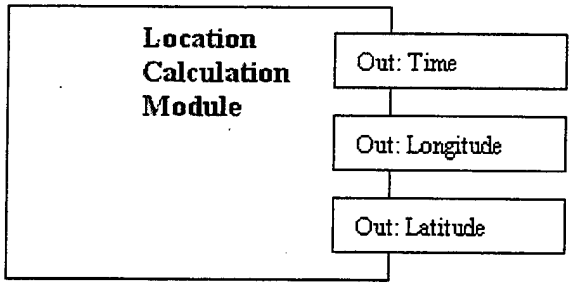


Figure 1. Typical External Interface of a Software Module

External interfaces are almost always well defined and well documented and usually include the type of the provided data, accepted values, and formatting information. This information is advertised as a contract for service and any other module or system can use this information if they have access to it. The software engineering paradigm of the Application Programming Interface (API) is one example of the use of external interfaces.

Continuing the previous example, the location calculation module may pass its information to a display module in the same system that has an interface that accepts a time, longitude, and latitude and displays the contact on a map. Figure 2 illustrates how such a system might work.

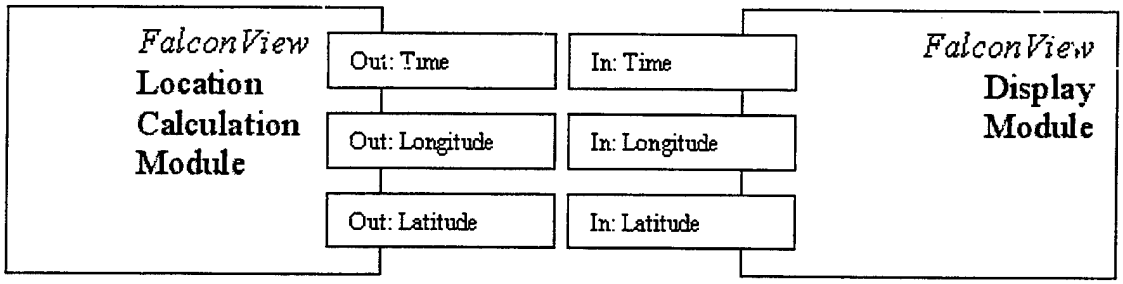


Figure 2. Interaction of Software Modules Through Interfaces

The ability to use external interfaces gives the system integrators access to the information being transmitted by systems. Unless the two systems to interoperate have

identical interfaces, however, issues of heterogeneity of meaning, and representation must still be resolved.

1. Reengineering Legacy Code

One possible way to integrate systems is to reengineer the system's source code and add in the desired changes to the interfaces. When integrating two systems a system integrator could choose to modify either system to accommodate the interface requirements of the other. For instance, expanding on the previous example, if a system integrator wanted to provide information from the *FalconView* location calculation module to an AFATDS display module that displays information using UTM coordinates, a programmer could find the code that calculates and exports the latitude and longitude. He could then add his own code to the legacy code translating the lat/long representation to a UTM representation and then make that view of the data available via the external interface. Figure 3 shows how this new interface might facilitate integration.

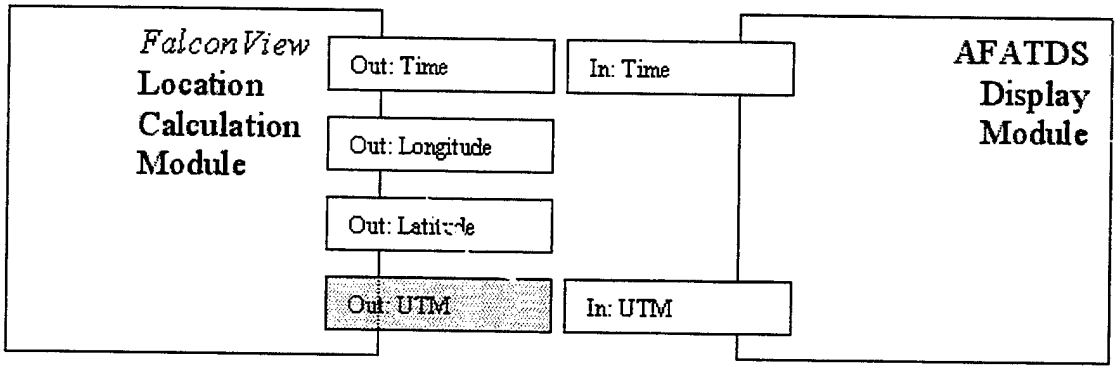


Figure 3. Reengineering Legacy Code to Match Interfaces

Unfortunately, the skills and time needed to find and change the legacy code that implements the system interfaces on a large scale, especially without automated assistance, has proved beyond the resources available to the DoD. A fundamental issue is that this solution requires domain experts for both systems to determine where the

systems differ in abstraction, meaning, and representation. Equally as time consuming is the act of programmers finding and modifying the code that exports the interface.

DoD system software that controls complicated equipment or provides special functionality is proportionally complex. Development of a Command, Control, Communications, Computers and Information (C4I) system, for example, can span several years and multiple contractors. The source code for the final application can run into the millions of lines. Unless the programmers who originally developed the system consciously built it with thought to changing or extending it in the future, and documented it as such, comprehending this code is a monumental task. Even if the original programmers were available, they would likely take longer to find the areas of the code that need to be changed, than to change the code itself. The Y2K problem, which cost the DoD an estimated $3.7 Billion, demonstrated the complexities and associated costs of reworking legacy software. For the DoD, the previously mentioned combination of Cold War pressures, security classification issues, rapidly changing software engineering practices, Moore's Law, and increased reliance on commercial hardware and software, mean legacy software will undoubtedly be difficult to understand, and even more difficult to integrate.

In summary, modifying legacy source code is often problematic. The code is often too complex to be easily understood and less sophisticated programming methods have left tightly coupled code which, when changed, may produce unexpected consequences in other parts of the application. In addition, for every new system added to the federation of interoperating systems, a new modification to the interface requires completely repeating this process. With each addition or modification to the legacy code,

the entire application must be re-tested and, possibly, re-certified. While this option provides system integrators a mechanism for interoperability, it is not an optimal method.

2. Software Wrappers

A more flexible way to integrate systems is through "software wrappers." Since software applications and modules communicate solely through rigidly defined interfaces, these are the only places where data representations need to be modified. Software wrappers, small programs that exist outside of the original legacy code structure, monitor a system's external interface and act as translators. Their only purpose is to translate from a system's original external interface to the desired external interface. They can exist on either or both the consumer and producer interfaces. Figure 4 shows a wrapper on the consumer translating the provided lat/long location data to a UTM representation suitable for AFATDS.

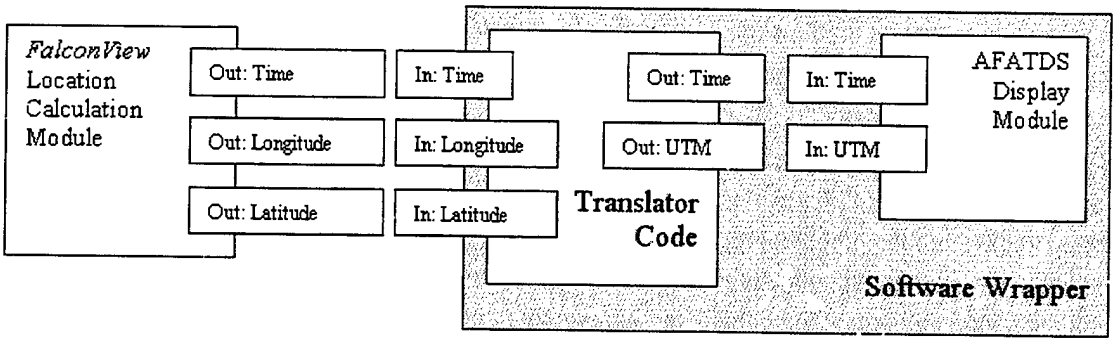


Figure 4. Use of Software Wrappers to Match Interfaces

If data representation difference is not an issue, wrappers may simply provide a mapping of data fields from the producer to the consumer. In all but the most simple of examples, this is not the case. Wrappers, however, can just as easily provide a data transformation capability. Since wrappers are stand-alone applications, any new change to the desired external interface only requires understanding the wrapper's code and then

modifying the translating and/or mapping functions to the new interface. Legacy source code for the wrapped application is never touched and thus testing for correctness and completeness or certification is only needed for the wrapper's code, instead of the entire application. Because the wrappers do not require complete integration with the legacy application, they can take advantage of the latest software engineering methods and tools.

Clearly software wrappers are a better way of doing business and, depending on the legacy application, may be the only option. For large-scale integration efforts, however, there is a scalability problem. For each additional system added to the federation, the programmer must write a new wrapper to communicate with every other system in the federation. Figure 5 shows **N** integrated systems where every system has the need to communicate with every other system in the network in a mesh-like topology. This will require writing N(N-1) or N^2-N wrappers. In large integration efforts, this solution quickly becomes combinatorialy infeasible.

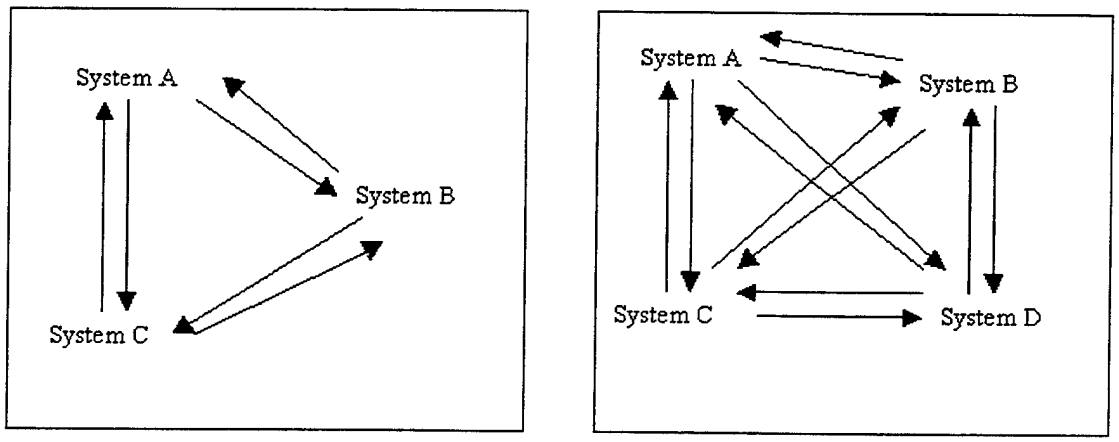


Figure 5. Pair-wise Software Wrapper Construction

3. Lingua Franca Automata

Lingua Franca – "Any hybrid or other language used over a wide area as a common or commercial tongue among people of different speech" [Web97]

One way to solve the problem of an exponentially increasing numbers of wrappers is to model the same federation of systems in a star-like topology. Every node that wants to communicate with another node must first translate to a common representation at a central node. This common representation can be thought of as the *lingua franca* for the network and may be a single system's representation or may be a "pure virtual" representation, used only for translation purposes. Using this methodology, every system now only needs two wrappers: one to translate from its own system specific representation to the common representation, and one to translate from the common representation to the system specific representation. For the identical federation of systems illustrated in the previous chapter, where every system needs to communicate with every other system, the number of software wrappers is reduced to 2N. Figure 6 illustrates this technique.

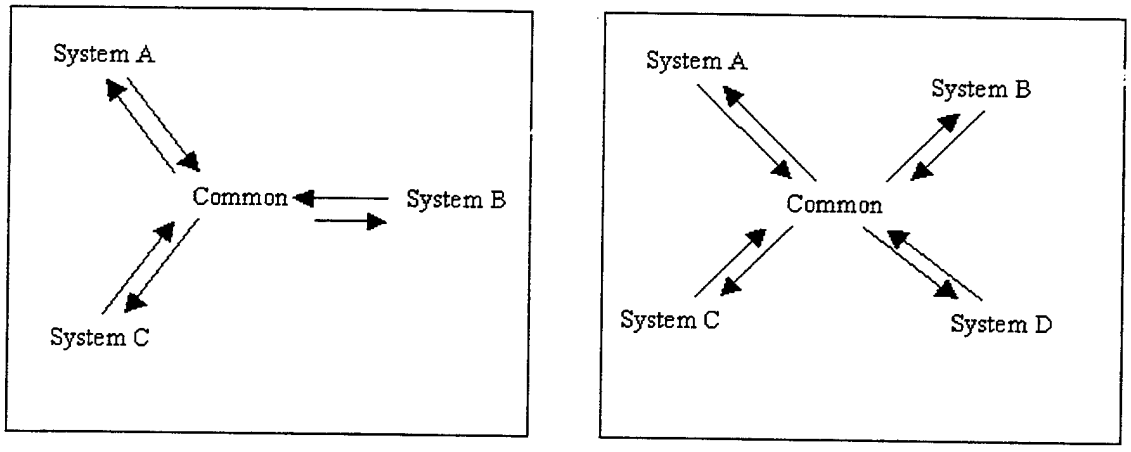


Figure 6. Benefits of Common Representation in Wrapper Construction

Before system designers can use a *lingua franca,* however, integrators must state or negotiate a common representation for all data types possible in the federation of systems. Deciding on an appropriate common representation is complicated when one considers the multitude of systems that may want to integrate. In almost all cases there will be some kind of data transformation (e.g. Latitude and Longitude to a Universal Transverse Mecator Projection (UTM) Grid Coordinate). In addition, there may be unexpressed information that is implicit in the system's original configuration, but after integration must be explicitly stated (e.g. accuracies, tolerances, uses, data sources). A functional model should allow system designers to advertise their produced data types, broadcast their desired consumable data types, and provide a mechanism for effortless representation translation somewhere between the transmitting and receiving end.

E. FEDERATION INTEROPERABILITY OBJECT MODEL

The Federation Interoperability Object Model (FIOM) is proposed as a way to model the multiple and diverse real-world entities systems either produce (export) or consume (import) through their external interfaces during system integration. The FIOM provides the *lingua franca,* or common representation, mentioned above. The FIOM also maintains system specific representations for every real-world entity and translation operation needed to convert a system specific representation to the *lingua franca* and vice versa. When implemented, it will provide a fully functioning, flexible way for system designers to integrate systems using state-of-the-art software engineering methods and technologies.

The FIOM is an instance of the generic Object-Oriented Model for Interoperability (OOMI) first proposed by [Young01]. Briefly, the OOMI extends the

contemporary object model class diagram to allow a robust and accurate description of system specific representations of real-world entities as well a way to resolve semantic heterogeneity. The use of a high level modeling technique, object oriented analysis and design (OOAD), traditionally used for software application development, allows the system integrator to leverage the research and automated tools that support this popular software engineering technique. Since the system entities, at various levels of abstraction, are treated as objects, the Unified Modeling Language (UML) can be used to provide a conceptual representation of the model. The OOMI includes the conventional properties of "name," "attributes," and "operations" and extends the conventional class diagram by the addition of a "class structure" property. This addition does not cause issue with any of the standard OOAD maxims such as inheritance, aggregation, or referential transparency. Figure 7, shows the contemporary object model class diagram, on the left, and the extended class diagram used in the OOMI, on the right.

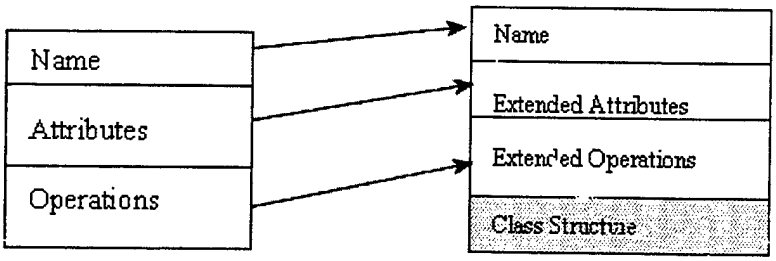


Figure 7. Modification of the Contemporary Object Class Diagram

As mentioned previously, extended classes provide encapsulation of all possible system-specific representations of attributes. Within the "extended attribute" property," the model can define the syntax (length, type, etc.) and the semantics (use, description, etc.) of every registered component system entity. Heterogeneity of representation is

handled by encapsulating translation operations to convert between the system view and the standard representation and vice versa for each attribute's system specific representation. For simple message passing, the OOMI only requires representations (and appropriate translations) for the sender and receiver.

Interoperability will likely include the exchange of data but may involve distributed processing as well. For a federation of interoperating systems, this means one system's required processes may be disseminated throughout the network to less critical or less heavily tasked systems for completion. Such a strategy would optimize the value of every system in the federation. We can accomplish this using the OOMI by assuming a function, regardless of platform, operating system, or programming language, can be treated as a mathematical rule mapping a domain (parameters) to a range (possible return values). These parameters and return values can be thought of as external interfaces for the procedure. As long as the appropriate translations are applied to the data being passed, interoperability is achieved. This is a natural extension of the message passing routine explained in the previous paragraphs.

The last type of heterogeneity addressed by the OOMI is heterogeneity of scope. User requirements dictate that each system in the federation has its own unique representation of real world entities' attributes (e.g. time in Local Mean Time (LMT) vs. Greenwich Mean Time (GMT)). In addition, these requirements may mean systems view the identical entity at differing levels of abstraction. For example, a personnel system may model a "person" in excruciating detail (e.g last name, first name, blood type, boot size, next of kin, etc.) while AFATDS may only care whether the "person" is a combatant or non-combatant. Differences in levels of abstraction are resolved through the class

structure property. This property captures individual systems' unique data requirements for a real-world entity; whether the system is producer or consumer, what attributes the system needs to consider **its** representation of the entity complete, and rules for handling missing, conflicting, or extraneous information.

The complete OOMI contains all the information about the standard representation of a real world entity as well as the specifics of each system's unique way of representing that entity. Translations are available for each system to allow conversion to and from the standard representation, and rules are defined to adjudicate scope issues. This OOMI class can then be used in the traditional OOAD means to compose other objects.

1. Building the FIOM

The first step in building a working FIOM is to identify a component system's entities. Most DoD information systems communicate using text based messages. In the past this has allowed the organizational change method of interoperability. Since messages were human readable, operators could transcribe information to non-integrated systems. This was not the original motivation for text-based messages, however. Among other advantages, text-based messages can have "tearlines," allowing the sharing of only limited portions of messages with allies or between systems of different classification levels. For this reason, and others, even systems designed and built today continue to use this method. Luckily, these messages are rigidly formatted and easily accessible. System integrators can treat messages as external interfaces for the system they wish to interoperate and use them to locate entities. For example, the location calculator from the

previous chapters might export a location (as part of a larger message) to another system using a message shown in Figure 8.

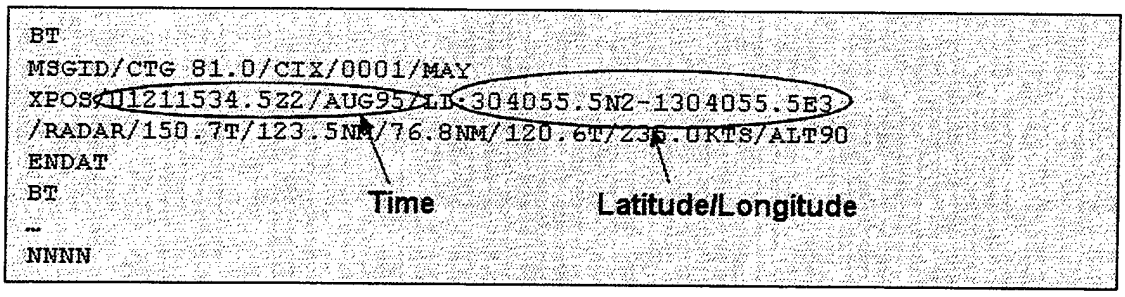


Figure 8. Location of Entities in Legacy Formatted Messages

The problem of locating the entities in messages is relatively straightforward since the messages, while human readable, are actually intended for automated systems. Parsing the document while following a set of message construction rules, allows data field recovery. Recently, this process has gotten even easier. In 1999, the Joint Technical Architecture (JTA), a mandate for DoD systems designers, directed that all domain- and application specific markup languages use the eXtensible Markup Language (XML) for tagged data [JTA00]. XML includes semantics along with the actual data, using tags in a hierarchical fashion meaning less ambiguity when locating and defining entities. Figure 9, on the next page, shows the same XPOS message presented earlier, using the new XML format.

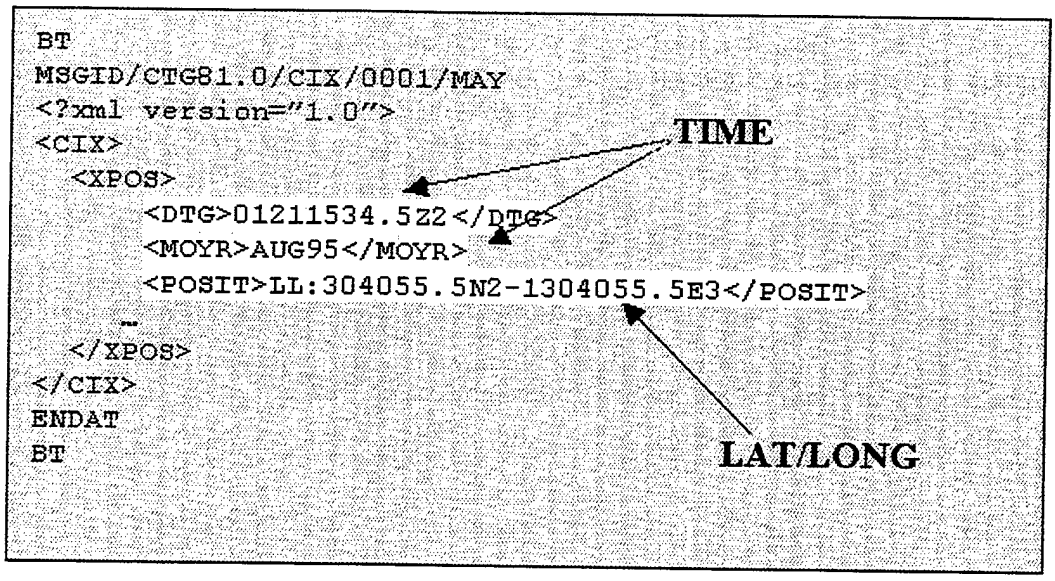


Figure 9. XML Representation of XPOS Message

Once the component systems' real-world entities are identified, the system designer can begin the integration process by matching his entities to those already in the FIOM. Even if his system is the first system to register with the FIOM, a baseline FIOM is available. This baseline is a collection of standard representations for common entities built using the Defense Information Infrastructure/Common Operating Environment (DII/COE) Namespace. These entities are used to instantiate the OOMI and create a FIOM consisting of individual Federated Interoperability Classes (FICs). FICs respect the OOAD paradigm and through composition and inheritance provide a domain-specific ontology. For example, a common notion in military messages is *POSREP* (position report). The portion of the baseline FIOM modeling *POSREP*, is shown in figure 10. *POSREP* is composed of three FICs *time, location,* and *unit name.* Each of these FICs may be a composition of less abstract FICs and may compose more abstract FICs (e.g. *unit_report*). This is transparent to the system integrator and translation operation will be available at each higher level of abstraction.

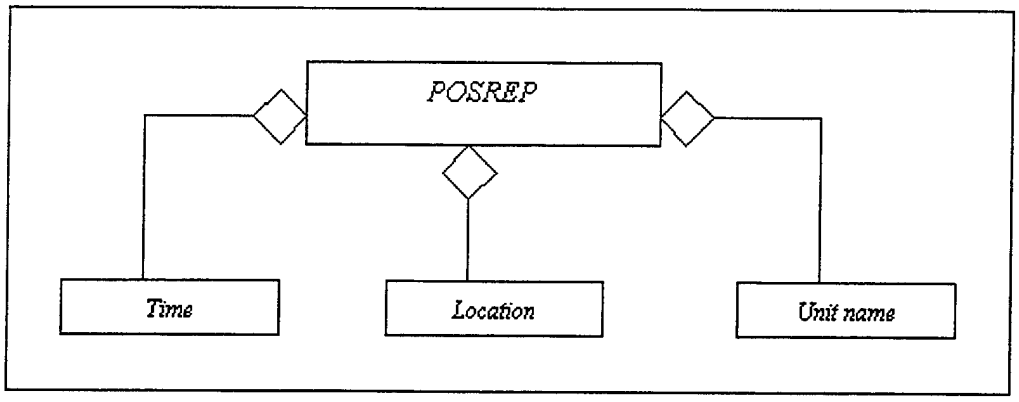


Figure 10. Composition of FICs

Once the integrator finds his entity in the FIOM, he has the opportunity to provide a software routine to translate from his type to the standard representation (for producers) or from the standard representation to his type (for consumers). Often, either the component system representation or the standard representation will describe an entity in additional detail (i.e. have more attributes). The integrator can solve this heterogeneity of scope circumstance by several means. If his system representation has additional attributes, he may discard them or, if they are semantically essential, he may extend the discovered FIOM entity to a new FIC that includes the additional attributes. If his system representation lacks attributes contained in the FIOM representation, he may define default values for the missing attributes, or choose to register a new FIC as a supertype of the discovered FIOM entity. He proceeds through all of his produced and consumed types until he has registered them all. If, during the registration process, he discovers his entity is completely unlike any in the FIOM, he may register it as a new FIC. This new FIC will be added to the baseline as the standard representation and the system integrator may compose it from already registered FICs, inherit from existing FICs or compose it from registered atomic attributes.

2. Using the FIOM

Once built for a specific federation of systems, the FIOM supports automated software wrapper construction. Since the FIOM understands entity specifics for both producer and consumer, seamless translation between disparate systems is possible. For example, when system A wants to pass a message to system B, system A will consult the model, find its representation of the entities it needs to pass, translate them to the common representation and then pass them along to system B. When the common representation of the entities arrives at system B, it will look through the model for the common representation, find the translations to its legacy format, and translate the objects to system B representation. As long as the source and destination are known by either system, this translation up to the bridge language and back down can be done at any point along the transmission path.

The FIOM can automatically easily construct a software wrapper since the system interfaces are in, or expect, XML format. XML Style Language (XSL) stylesheets provide a way to convert from one XML representation to another. This process may involve an intermediate step (converting from component system XML to common representation XML) or may be done directly to the destination system's XML representation. The separation of data from presentation information using XML also means that future extensibility is provided for by additional stylesheets. For example, the external interface from AFATDS in XML format can provide contact information to *FalconView* using one XSL stylesheet, construct a human-readable Adobe Acrobat .pdf file using a different stylesheet, and place the information into a common contact

database using a third stylesheet. A typical message passing arrangement is illustrated in Figure 11.

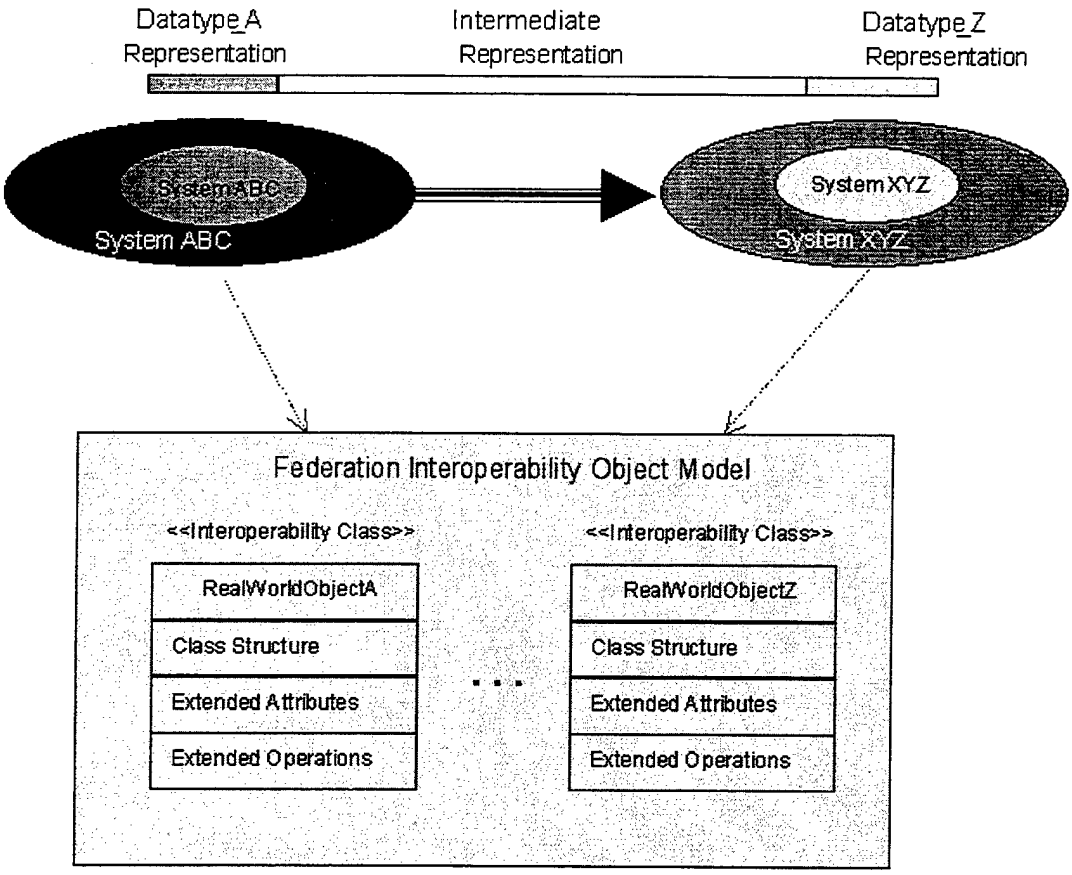


Figure 11. Message Passing Using the FIOM From Ref. [Young01]

F. SUMMARY

The FIOM is an instantiation of the OOMI for a specific federation of interoperating systems. The OOMI is a generic abstract model of real world entities that provides mechanisms for preservation of individual systems' attribute representations, operation semantics, and class structure. Because the OOMI respects the OOAD paradigm, OOMI classes can use all the modeling techniques and automated tools used

for OOAD including, but not limited to, inheritance, composition, aggregation, and UML representation.

A baseline FIOM exists for application specific domains. This baseline is built using the DII/COE Namespace Registry to help ensure that the common representation is as close to the DoD standard as possible. This FIOM consists of real world entities at various levels of abstraction allowing system integrators to match their entities at the proper level. These entities are modeled using classes called FICs.

When system integrators register their systems with the federation, they attempt to match their system objects, contained in the system's external interfaces, with real world entities in the FIOM. If a match is found, the integrator may include a software transformation from the system specific representation to the common representation and vice versa. If a match is not found, the integrator may add his entity to the FIOM as a new FIC.

Once the system integrator has registered all his entities, he uses the FIOM to automatically construct wrappers for his system to communicate with any other system in the federation. Wrappers use system-built XSL stylesheets that translate the XML formatted external interfaces of the systems from one representation to another. This procedure can take place anywhere along the message's transmission path.

III. CORRELATION METHODOLOGY

A. BACKGROUND

In order for systems in a federation to share information, system designers must first agree upon *what* they are sharing (differences in semantics) as well as *how* that information is to be represented (differences in syntax). In chapter II a general model for information sharing, the FIOM, was described. The FIOM provides a way to define a common vocabulary for describing the entities in the federation (an ontology), a methodology for composing individual entity attributes into more abstract ideas (Object Oriented Design), and a way to store system-unique entity representations (the extended contemporary object class diagram). Use of this model provides improved heterogeneous system information exchange by providing semi-automated wrapper construction via translation to the common representation and back to the destination system representation where commonalities exist. In order to use this powerful model, however, system designers must determine where their particular system entities fit into the hierarchy of FICs defined by the FIOM. The process of discovering entity correspondences between component systems, or between component systems and the standard representation, and then including the system's representations in the model is referred to as **discovery**.

B. WHAT CONSTITUTES ENTITY CORRESPONDENCE?

Early attempts to integrate disparate systems were done strictly by hand. Although time consuming, in one instance four hours per data element, the engineers did a good job of deciding when two objects in different systems represented the same real-world entity [LC94]. Interestingly, automating the natural human capability to say two

objects from different systems represent the same entity is fraught with difficulties. In general terms, what system designers are trying to determine when they match two systems' entities is the *similarity of semantics* between two objects.

[Wood85] defines *semantics* as the scientific study of the relations between signs and symbols and what they denote or mean. Obviously, there is no way to exhaustively capture the semantics of an entity in a system. Instead, system designers selectively choose those elements that sufficiently describe the meaning or use of the object that will be most profitable for the intended use of the system. For closed systems, this is not a problem. The assumptions made by engineers about what is semantically important when describing data elements are consistent with other parts of the system that utilize that information. When these systems interoperate, however, the same object may appear in two different systems with two different meanings or the same object may be represented differently in either system.

Sheth and Kashyap [SK92] offer database integrators a taxonomy for semantics based on various levels of "sameness." Levels of correspondence include including "semantic equivalence," "semantic relationship," "semantic relevance," and "semantic resemblance." While it might be useful to know the level of semantic correspondence between two objects, for interoperability, the only criterion that seems applicable is "means the same." Even this semi-formal definition can cause problems, however. The "means the same" criterion was used by Clifton when he conducted a large database integration effort for the Air Force in 1994 [Clif97]. Surprisingly, he found that when researchers provided domain experts with a relatively small set of possible attribute correspondences and asked "do objects A and B mean the same thing?," domain experts

found the question ambiguous. Even worse was the question "do objects A and B represent the same real-world entity?" The discomfort shown by domain experts was partially because they were not intimately familiar with **all** systems being considered for integration. Since the scope of representation for entities varied depending on the entity's use and context, making generalizations was neither possible nor desirable. Clifton found it better to pose his question as one of data interchange suitability. The question was changed to "ignoring representational differences, if System A sends you a missionStartTime, can you use that in System B in place of takeOffTime?" Domain experts found this question answerable.

For integration efforts where data is exchanged between disparate systems (which includes our system for assisting this effort), perhaps the best measure of semantic correspondence is substitutability. When registering with the FIOM, engineers ultimately determine entity correspondence by using criteria similar to the software engineering Liskov Substitution Principle [Liskov87]. When applying this criterion, however, one must keep in mind whether the provided entity will be produced or consumed. For producers, if their exported entity can be substituted for the FIOM real-world entity, then there is entity correspondence. For consumers, if the FIOM real-world entity can be substituted for the component system's representation, then there is entity correspondence. The opposites of these two statements are not always true. Strict adherence to this policy allows a system to establish entity correspondence without *a priori* knowledge of another system since both know the FIOM representation.

C. CORRELATION EFFECTIVENESS

The necessity for automation of correlation in Very Large Databases (VLDBs), software component libraries, and heterogeneous system integration efforts is due to extremely large search spaces. Depending on the amount of data to be recovered and the size of the database, simply performing the necessary comparisons can take a prohibitive amount of time. A search algorithm that attempts to limit the search space must provide likely matches while still guaranteeing some level of assurance that possible matches are not prematurely discarded. These competing priorities for correlation algorithm designers are detailed in [SM83] and [Stei91]. Salton and McGill provide six evaluation criteria for determining the effectiveness of information retrieval systems; precision, recall, effort, time, presentation, and coverage. All six are key to the correlation of similar entities within the FIOM.

Precision is the ratio of relevant entities returned to the total number of entities returned. In a perfect system this number will be one since the perfect system will only return exact matches.

Recall, on the other hand, is the ratio of relevant entities returned to the actual number of relevant entities in the search space. High recall ensures that you have not set your threshold for correctness so high that possible matches are discarded.

Effort is the amount of physical and intellectual work needed to perform the correlation.

Time is the measure of how long (either in CPU epochs or real elapsed time) it takes to perform the correlation.

Presentation is the method that the query uses to present data to the user. Different types of presentation may include a ranked list, where unlikely matches are included, or a single "best match" candidate.

Coverage is the measure of the search space the algorithm is able to correctly query.

The effectiveness of a search, measured by the criteria listed above, usually depends on how the data to be searched is organized (representation) and how the querying algorithm is written (search). [Stei91] notes that there is a trade-off in the amount of effort spent organizing the data and the effort designing a search algorithm. Search data logically and rigorously organized may require a relatively unsophisticated search to achieve effectiveness. Unstructured data will require more effort in algorithm design to achieve the same level of effectiveness.

D. CONVENTIONAL SEARCH METHODS

The explosion of the Internet and an increased interest in reusing software components has led to major advances in search algorithms. Both of these domains have unique and significant challenges for information retrieval. A short examination of these techniques illustrates some general methodologies that will be applied when attempting to correlate entities.

1. Browsers

A *browser* is a general purpose, usually window-based tool for looking through collections, categories, or hierarchies of components at various levels of abstraction [Meye88]. In the Internet domain, a search engine such as *Yahoo!* which uses hierarchical topic headings is an example of such a system. This type of search is a

logical choice for an object-oriented hierarchy where inheritance is heavily used (e.g. Java's *Object* base class). One disadvantage is that users of this kind of system must be intimately familiar with both the entity they want to correlate as well as the overall structure of the structure browsed. The only way to retrieve a potential match is to navigate through the tree structure down to the exact entity. Once an entity in the search space is noted as a possible match, the only way for the integration engineer to assure this is to physically inspect its attributes and operations. Pair-wise correlation of the returned entity and the component system entity may itself be time-prohibitive.

2. Keyword Search

In a keyword search, the user provides a list of words he thinks describe the entity he wants to correlate. The search algorithm then uses these terms to retrieve entities in the search space that share similar keywords. Improvements to the general algorithm allow Boolean expressions (AND, OR, NOT), constraints, and synonym replacement as well as ranking the importance of search terms by proximity. A search engine such as *AltaVista* is an example of this kind of search.

Anyone who has searched the Internet using this kind of search engine understands the shortcomings of the algorithm. Although conceptually easy to use, the precision and recall of the search are determined by the quality and number of the keywords provided to the search engine. A small number of keywords, even perfectly chosen, may mean high recall but low precision and a set of possible matches that cannot be intelligently presented for subsequent search methods. A large number of keywords may result in high precision but low recall.

3. Faceted Classification

[Prie85] addresses the problem of queries constructed from an unrestricted vocabulary by introducing the notion of "facets" for classifying entities. Facets describe a broad classification of entities or actions and have within them a list of terms that apply solely to that facet. Each entity in the search space is classified using this constrained vocabulary. Queries are formed from terms in the facets of the search space and are easily resolved against indexed possible matches.

The major disadvantage of this method is that it relies on a librarian to catalog the search space. This itself is a time-intensive process. Also, the person submitting the query must perfectly agree with the librarian on how to classify their particular entity for high precision. Prieto-Diaz mentions how partial matches can be resolved and ranked using a thesaurus and a conceptual distance graph to evaluate semantic "closeness." Regardless, like browsing, this method requires the integration engineer to have an unrealistic understanding of both his entity and the structure and semantics of the search space.

4. Semantic Integration (SEMINT) Tool

One difficulty common to all of the traditional search methods is that they rely on metadata to describe themselves to the world. Sometimes this metadata is explicit; the facet terms chosen by a librarian to classify an entity. Other times the metadata is implicit but is extracted, either automatically or manually, to form a definition of entity; a list of common words from an html document indexed for a keyword search. If the metadata is explicit, building the repository of metadata is a time-intensive process. If the metadata is implicit, the precision is strictly determined by the quality of the

algorithm used to build the search index. Construction of queries suffer from the same shortcomings. Traditional search methods use syntactic information to build a semantic picture of the entity. For non-text-intensive entities, this syntactic information is often minimal.

Research into database integration by [DKM+93] pointed out that semantics are located in the database model, conceptual schema, application programs, and the minds of users. Automation of resolution of semantic differences without excessive reverse engineering procedures, unrealistic knowledge of how the information will be used, or intensive human oversight implies automation can only rely on the first two. From the conceptual schema one can determine the entity names, field specifications (length, acceptable values, data type), and field constraints. If this information is not available in the schema, it can often be determined by examination of the data contents. Close examination of the data contents also provides statistical information about **actual** data values and domains. The combination of this information provides insight into semantics for database integrators. Used with other techniques it provides discriminators for semantically matching attributes in disparate databases.

Semantics-based searches attempt to use the actual behavior of entities to establish correlation. The ability to capture an entities' dynamic behavior and compare it against other entities' behavior promises a better answer to the essential question of "do A and B mean the same thing?". As pointed out earlier, these semantics will aid integration engineers in determining whether entity A can replace entity B for interoperability. The presentation of this information is such that it is not mutually

exclusive with traditional methods and can be used in conjunction with other search algorithms in a multi-step filtering process.

Li and Clifton propose semantic integration is possible using artificial intelligence techniques [LC93] and automated software tools. The semantic integrator (SEMINT) tool was built to aid in heterogeneous database integration. Entity resolution in this domain has specific requirements due to the large search space, anomalies due to one-word field names, lack of textual descriptions of each field, etc. To mitigate these problems, SEMINT builds a list of discriminators and their values for each entity from the database conceptual schema. Additional discriminators are determined and values assigned, based on the data contents of the fields in the database. The combination of this information defines an entity by its type definition and actual values.

Queries in a pair-wise manner using this extensive data would be both time and effort prohibitive. To allow the best chance of matches, the database representing the search space is indexed using a trained neural network. The most significant discriminators across the entire database are chosen and each entity's discriminator values are provided to a neural network. The network is then trained so that when the reference database's entity's discriminator values are provided, the network outputs the reference database's entity's name.

When a database integrator wants to determine if attribute A of his database exists in the reference database, he provides the previously trained network with attribute A's discriminator values. Based on the output of the previously trained neural network, a list

of possible matches is provided. This process continues for each attribute in the integrator's database.

The SEMINT process is discussed in detail in chapter IV of this thesis and forms the basis for Young's interoperability tool correlation algorithm.

D. SUMMARY

Querying a large set of entities and retrieving probable matches with high recall and precision is an area still actively under research. The problem for all domains is essentially the same: how do you semantically match a user's entity to what is already in the search space?

Traditional search methods are based on syntactic information. In the search space this information is provided via metadata either extracted from the established entities or added by hand by a librarian. This may be a time-intensive and subjective effort or may be done by an imperfect parser. Either way, the results of a search will be determined by the skill of the user making the query.

One potential solution to the problem of semantic matching is the SEMINT tool. By gathering information about the restrictions on data as well as statistical information about actual data values in a representative set, the tool builds a representation of what a specific entity should look and act like. The reference data is used to train a neural network which is then used to determine whether a data representation from another database is like that in the reference database. This tool does not require programmed heuristics, excessive user input, or unrealistic amounts of metadata.

IV. FEDERATION INTEROPERABILITY CLASS CORRESPONDENCE

A. ASSUMPTIONS

The FIOM was conceived as a generic model for interoperability that would leverage the advantages of object oriented analysis and design currently used in software application development for enabling interoperability among legacy DoD information systems. Currently DoD systems pass information in preformatted text messages with data fields separated using an agreed upon text delimiter or through fixed length formatting. This provides a human readable message that can also be automatically parsed and the data in the message used within system applications.

While the current form of message passing is sufficient, increasing interoperability of heterogeneous systems and a desire to extend the scope of produced data has led system designers to look for a better solution. A solution that is easy to implement is to format messages using the extensible Markup Language (XML). XML provides the recipients of messages access to additional information about the contents of the message by including tags around the provided data. In the simplest wrapper implementation, as outbound messages are constructed, the field names for each data element are included along with the data itself. XML's flexibility in unrestricted tag names, tag nesting, and data representation allows its application across almost any domain. This practice was covered in some detail in chapter II of this thesis.

While the FIOM is applicable to some domains, it is not applicable to all. To limit the scope of this thesis, some assumptions were made concerning the types and

structure of data expected, the documentation conceivably available to system integrators, and the experience and skill of users of the IDE.

1. Existence of Detailed XML Schemas

One advantage of XML is that system designers can impose strict rules concerning the construction of the XML document itself. While XML documents can be freeform, they must always respect the XML structure. A *well-formed* XML document means that the tags are properly nested and the XML syntax adhered to. For example, each opening tag must have a closing tag and any tags opened below it in the tree must have a closing tag before the first tag can be closed.

There is also a more rigorous method for ensuring an XML document exactly meets the system designer's intentions. This is the XML schema. An XML schema defines the physical structure of the XML document and sets rules for how tag names can be used, cardinality, and relationships between tags. A schema can also dictate what datatypes are legal values for elements. The World Wide Web Consortium (W3C) has developed a list of primitive data types that allow schema developers to dictate to XML message producers both the physical schema and data contents of the message. When an XML document is parsed, the parser will automatically check the contents of the document against the schema and will throw an exception if they do not match. An XML document that correctly meets the schema is called a *valid* document. A comprehensive XML schema is similar to a database physical schema and provides our system the same semantic and syntactic information database integration tools leverage.

Any system that wishes to integrate with other systems using the FIOM must have a detailed XML schema for each message passed. This assumption is reasonable

for several reasons: First, the ability of XML documents to completely represent both primitive data types as well as complex data structures means that system designers can completely represent their legacy system's existing external interfaces; Second, the overhead of checking the integrity of external interfaces is carried out in the XML parser if a schema is available; and last, for the reasons mentioned above, the DoD is already replacing older text messages using XML. The ability to validate an XML message against its corresponding schema more than justifies schema use. It is therefore in the wrapper designer's best interest to use XML's automated checking to ensure a consistent, complete, and correct external interface as well as to enhance interoperability. Our correlation algorithm will not require any more detailed information than should already be readily available. An example of a detailed schema is presented in Appendix A.

2. Domain Expert Participation

No integration tool can perfectly predict semantic correlation nor should we expect it to. The current method of system integration relies almost completely on domain experts evaluating both the producing and consuming systems to determine object correspondence. This is a time-consuming and imperfect method. The FIOM Integrated Development Environment (FIDE) will not automatically perform object correspondence. Its purpose is to limit the amount of pair-wise comparisons system integrators must evaluate by hand by resolving semantic and syntactic similarities between systems and the FIOM. The final determination on whether an object from the component system's external interface matches that in the FIOM is made by someone intimately familiar with the use of the data in the component system. This solves the "Do A and B *mean the same*?" dilemma encountered by Clifton in [Clif97].

B. OVERVIEW OF CORRELATION PROCESS

Object correlation using the F-IDE is done using a multi-stage query. The first stage is a full-text correlation. The second stage uses the SEMINT technology covered at the end of the previous chapter. Since the SEMINT tool is processor intensive, the F-IDE attempts to limit the number of classes compared in the second stage by only using the results of the syntactic comparison stage. The user is presented with ranked results for each stage and has the ability to pass any or all of the FIOM classes to the subsequent stages.

To start the process the user must only supply the F-IDE with the filename and path for his entity's XML schema. The tool then processes the component system's schema and extracts selected keywords that convey information about the semantics of the entity. This list of keywords is then compared to the keyword list stored for every FIC standard representation and registered class representation. The user is presented with a ranked list of possible matches based on syntactic information (i.e. similar keywords in both lists). As syntactic information does not always completely or sufficiently define an entity, no class is discounted. Instead the list of every entity representation in the FIC is presented sorted by scores from the syntactic search. This allows the user to selects those FICs with high syntax scores to the second stage of the query. A screen capture of the tool after the first stage comparison is shown in Figure 12. Note that the user has chosen to include only entities with a syntactic score of 91% or better for the second stage of the correlator.

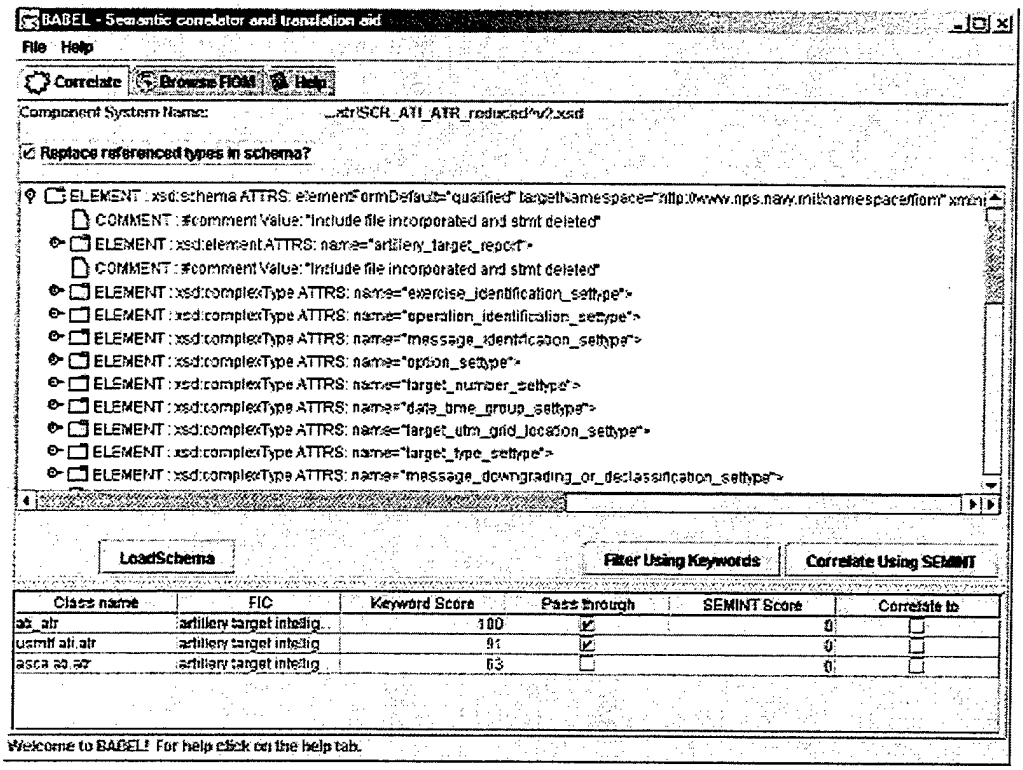


Figure 12. Correlation Tool Showing Results of Syntax Match

After deciding which FIOM classes to continue comparing with the component system entity, the user invokes the second stage of the correlator. In this stage the F-IDE compares the component system entity's attributes against those of the FIOM classes using the SEMINT tool. The results of this stage are presented to the user as a score for each FIOM class. This ends the automated correlation effort. The user can then use these "likely matches" to manually register his entity with the FIOM.

C. NORMALIZING AND INDEXING THE SEARCH SPACE

Before a user can attempt a query against the FIOM, the registered classes must be normalized and indexed to reduce the expended effort, and maximize the precision and recall of the search algorithm. This is done sequentially at time of registration of new entities and is completely automated by the F-IDE tools. Regardless of whether the

FIOM registers a FIC from a federation ontology such as the DII/COE Namespace Registry or a component system, that class will be designated as the standard representation for the entity. Any subsequent component systems entities that register will have the possibility of also representing this entity. The size of the FIOM does not change the overall normalizing and indexing methodology. This behavior is desirable since the model then allows comparison of two component systems or two external interfaces of the same component system if the system integrators wish to use it that way.

1. Gathering Syntactic Information

In chapter III, the relative merits of various search algorithms were discussed. Traditional search methods rely solely on syntactic information and work by comparing syntactic metadata between documents. Since the second-stage SEMINT tool does not preclude their use, we can use text based searches to filter the number of candidates for more time-intensive semantic matching processes. This will reduce the overall amount of effort needed to search the FIOM using the SEMINT tool.

Metadata for component system entities exists in various locations depending on the system implementation. We assume that the XML schemas will be built from component system data dictionaries. Data dictionaries are typically documentation of a system's external interface to assist in message preparation. For older systems this information might include the field delimiter character or the field lengths to assist in hand construction or deciphering of text-based messages. For most systems the data dictionary will describe the field's name, purpose, relationships to other fields, and possible legal values. The XML schema encapsulates this information in element names, attributes of the element tags and the data contents of certain schema elements.

Once a FIC is created from a FIOM baseline model (DII/COE Namespace Registry) or a component system is registered, the F-IDE will first parse the corresponding XML schema file and build a keyword list. This list will be used during the first stage of the F-IDE correlator. The text inserted into the keyword list is drawn from the schema fields that encapsulate the data dictionary entries deemed to convey syntactic information about the entity. The list of schema fields used is shown in Table 2. Since the schemas never change once registered with the FIOM, the FIDE saves the keyword list to a separate XML file for use during the discovery process. An example of a keyword list is included in Appendix B.

Field	Attribute	Details
xsd:element	"name"	The name attribute typically equates to the field name used in the underlying database.
xsd:element	"type"	For schemas using global types. This attribute's value is usually descriptive of the kind of data in the subtype. (e.g. "date_type")
xsd:documentation	N/A	The text in this element is the "description" field from the data dictionary. It is typically a human-readable free text explanation of the field's use or format.
xsd:attribute	"name"	Gives amplifying information about a simple or complex type.
xsd:enumeration	'value"	Used to constrain the values of types. Usually used to limit a message field to several values which will reveal the use of the message (e.g.. "SUB", "SURF", "AIR")

Table 2. List of XML Schema Fields Used for Keyword Determination

2. Determining Entity Semantics using SEMINT

As discussed in chapter III, SEMINT uses a neural network to determine likely attribute correspondences. Before we can use the FIDE SEMINT capability for entity correspondence, however, we must train and store neural networks for the FIC's standard representation and registered component class representations.

a. Building Representative Vectors

In the database version of the SEMINT tool, the discriminators chosen are readily available and can be automatically discovered from the schema and the database contents. These discriminators include, but are not limited to, field length, data type, constraints, and allowable values. The F-IDE mimics the database information gathering procedure by parsing the XML schema and building vectors of normalized values for each atomic attribute. A vector based on schema information and values will represent each attribute in the entity (XML Schema *simpleType*). Some values are simple binary values, such as "mandatory" or "optional". Other values, such as field length or minimum occurrences, are mapped to a range [0,1]. Data types and other category information require vectors of binary values vice single values. Merely assigning an arbitrary value in the range [0,l] for each data type would construe one data type is "closer to" one type than another. For example if *string* was represented by 0.1, *integer* by 0.2, and *float* by 0.3, the SEMINT tool would deduce that a *string* is more like an *integer* than a *float*. This is clearly incorrect and so, instead, each type has a vector with a unique value. In our case *string* is represented by the vector <0, 0, 0, 0, 1>, *integer* by <1, 0, 0, 0, 0>, etc. Since each element of these vectors will stimulate its own neuron in a neural network, there is no presumed "closeness" between data types. Missing discriminators in the XML schema are handled by mapping zeros in the vector. This does not prevent use of the neural network but will lower the precision in the query results.

Like the syntax information used for the keyword map, all the information needed to build the vectors is located in the XML schema. Parsing it does not require

user supervision or training. A list of the discriminators used by the F-IDE SEMINT tool prototype is presented in Table 3.

Field	Attribute	Value to vector
xsd:element	"minOccurs"	If 0 (optional) - 0 Otherwise – 1
xsd:element	"maxOccurs"	If not stated – 0, 0, 0 If 1 – 0, 0, 1 If 1 < x < unbound – 0, 1, 0 Otherwise – 1, 0, 0
xsd:restriction	"base"	If string – 0, 0, 0, 0, 1 Date – 0, 0, 0, 1, 0 Boolean – 0, 0, 1, 0, 0 Decimal – 0, 1, 0, 0, 0 Integer – 1, 0, 0, 0, 0
xsd:length	N/A	Normalized to [0,1]
xsd:pattern	N/A	Has pattern defined – 1 Otherwise – 0
xsd:minLength	N/A	Normalized to [0,1]
xsd:maxLength	N/A	Normalized to [0,1]

Table 3. XML Schema Fields Used for Discriminators.

Figure 13 illustrates how vectors of discriminators are built using the date_time_group attribute as an example. This attribute appears at least once, at most once, is a string, has a length of nine characters, has a distinct pattern defined, has minimum length of nine characters, and a maximum length of nine characters.

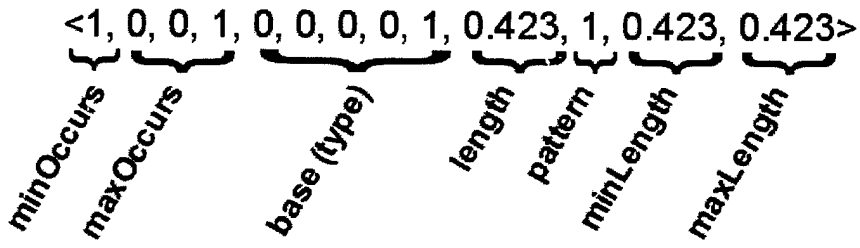


Figure 13. Discriminator Vector for date_time_group Attribute

b. Clustering Attributes

Once each attribute is assigned a representative vector, these vectors are provided to a self-organizing map (SOM) algorithm (please see [Koh87] for details) to classify them into "clusters" of like attributes. A SOM consists of a two-dimensional output grid fully connected to an input signal. After unsupervised training, the SOM will define a non-linear mapping from the inputs to the output grid. This step is necessary because we cannot guarantee a unique vector for every entity attribute. SEMINT uses a neural network to map input patterns (an attribute's vector) to an output pattern (the attribute itself), and if we do not cluster identical or nearly identical input patterns, the neural network will either be un-trainable or will not consistently output the correct output pattern due to ambiguous cases. Once the SOM clusters similar attributes discriminators together, we can use the neural network and present the cluster's member attributes to the system integrator to resolve.

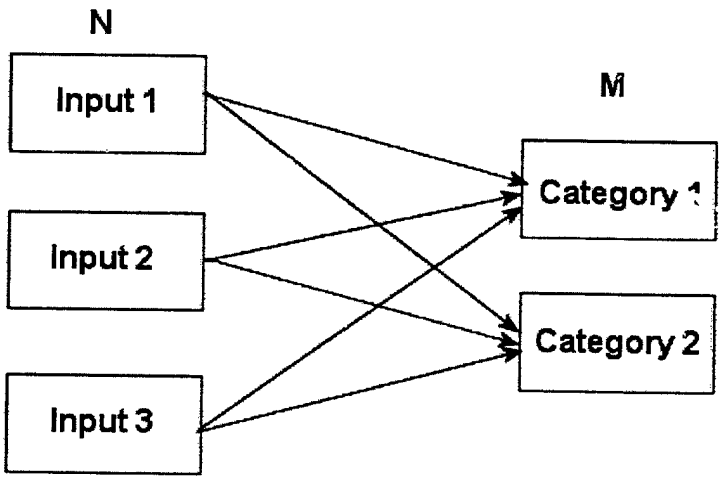


Figure 14. Self-Organizing Map

A typical SOM is shown in Figure 14. The N input nodes are the values from the attributes' discriminator vectors and the M output nodes represent the number of dissimilar attributes in the system entity. A SOM usually requires three parameters: a learning rate (also called *alpha*), a radius, and M. In the F-IDE SEMINT tool *alpha* and the radius are pre-determined and therefore the only missing input to a self-organizing map is the value of M. This is automated as well. The F-IDE SEMINT tool conditionally chooses M by comparing all attributes input vectors and only considering an attribute "dissimilar" if that attribute's vector differs by at least one value from any other attribute's vector thus eliminating the worst-case scenario where two input vectors are identical but represent different attributes. The self-organizing map then uses an unsupervised learning algorithm to cluster like attributes into M categories. Each of these clusters is then assigned a "cluster center weight" that represents that category of attributes with an average discriminator vector. These vectors allow the calculation of degree of similarity for output values from the neural network and will provide the input neuron values when training the neural network.

c. Training the Neural Network

Neural networks are traditionally used for pattern recognition. A back-propagating neural net is first trained using supervised learning unit it successfully maps all neuron stimuli values to desired outputs within a given threshold. We can then apply uncategorized neuron stimuli values to the trained network. The output will be one of the categories used during training or a quantity representing the closeness to one of those categories. For the F-IDE SEMINT tool, a back-propagating neural network is constructed using N neurons representing the discriminators chosen from the XML

schema and M output nodes, where M is the number of clusters (or categories) built by the self-organizing map. Only one middle, or hidden, layer is built of (N+M)/2 nodes as this is typically sufficient for back-propagating neural networks. A typical neural network is shown in Figure 15.

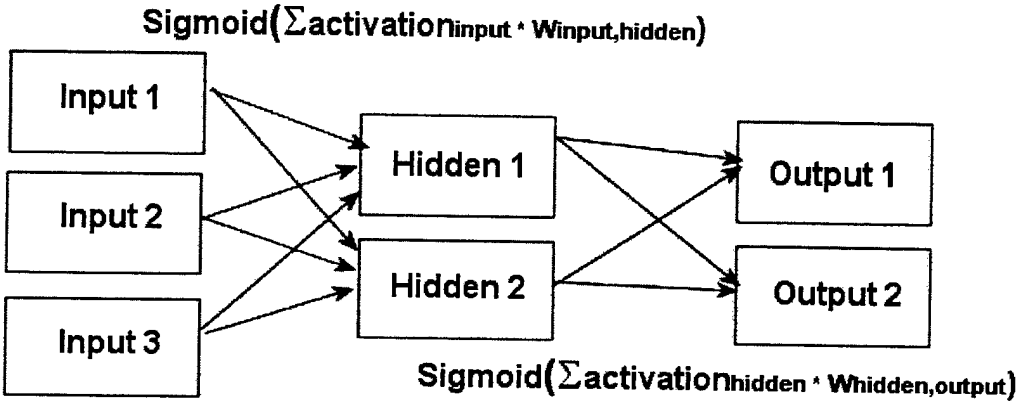


Figure 15. Typical Back-Propogating Neural Network

The network is trained by initially assigning nominal values to the edge weights. Training data, in our case the attribute vectors, is then presented to the network and output values noted. If the difference between the desired output and the expected output is not within a set tolerance (hard-coded for our application), the edge weights are adjusted; first the hidden-to-output edges and then the input-to-hidden edges. Once the error is reduced below the threshold value, the network is considered "trained" and it is stored for use in the discovery process.

To illustrate how the trained network will perform, suppose we were training the neural network with our previous "date_time_group" example. Suppose the message entity had four attributes, *date_time_group, sender, recipient,* and *declassification.* The output of the neural network is easiest to represent by a vector

representing the result node values, which equate to the category of the attribute presented. *Date_time_group* is represented by <1, 0, 0, 0>, *sender* by <0, 1, 0, 0>, etc. After passing each of the four attributes' vectors to the neural network we train it until it responds with the correct attribute (within the error threshold). For example, if we present the network with <1, 0, 0, 1, 0, 0, 0, 0, 1, 0.423, 1, 0.423, 0.423> (the *date_time_group* attribute's discriminator vector), the network will respond with <1, 0, 0, 0> (the *date_time_group* attribute category).

D. PERFORMING OBJECT CORRELATION

The trained neural nets stored for each registered class are used in the second stage of the correlation. In order to use these networks, the component system entity's attributes must be mapped into normalized vectors in the same fashion as the FICs. The component system's schema is parsed and the vectors of discriminators built. These vectors are then presented to the input neurons of each registered entities' trained neural network and the output values noted. The output vector for each component system attribute is then compared with the cluster vectors for the every one of the neural network's actual attributes. A single floating-point number is assigned for this pair-wise match based on the degree of similarity between the component system's attribute output vectors and the FIC's attribute cluster vectors.

Once every component system attribute has been compared against every registered class attribute, an NxM matrix is formed from the resulting pair-wise comparisons (Figure 16) where N is the number of component system class attributes and M is the number of registered class attributes. In the figure α, β, γ, and δ are system attributes and a, b, c, and d are registered class attributes. Light gray lines represent the

likely attribute matches but system integrators will ultimately determine this by manual inspection of attribute data type definitions. A single floating point value is assigned for each class by taking the highest SEMINT score for each component system attribute, and taking the square root of the sum of their squares (Figure 17.). This will effectively tell the user how much, in general, the two classes are alike. The system presents this number to the user as the score from the SEMINT stage of the correlator for each registered class in the FIOM. From the example shown in Figure 16, the score presented to the user would be $(.52^2 + .48^2 + .99^2 + .76^2)^{1/2}$ or 1.435. This number will gain meaning when compared with the results of other class comparisons.

	a	b	c	d
α	.52	.20	.37	.08
β	.05	.48	.19	.15
γ	.13	.12	.04	.76
δ	.24	.11	.99	.03

Figure 16. Matrix of Attribute Comparisons Using SEMINT Tool

$$\left(\sum x_i^2\right)^{1/2}$$

Figure 17. Calculating the Entity Average SEMINT Score.

Once the SEMINT tool returns its ranked results, the system integrator can use those marked as most likely to have the same semantics to verify and validate the correlator. This process of defining the links and conversions between entities is noted as an area of further research.

E. SUMMARY

The F-IDE attempts to correlate component system entities with those previously registered in the FIOM by a two-step process. As systems are added to the FIOM their XML schema files are parsed and keywords are extracted and saved. Additionally, semantic information from the schema is used to train a back-propagating neural network until it maps each attribute's unique information (the discriminator vector) to the attribute itself. Similarities between attribute discriminator vectors are resolved by organizing like attributes into categories using a self-organizing map.

When a system integrator wants to correlate an entity with a class previously registered in the FIOM, he provides the schema to F-IDE. The schema is parsed and its keywords compared with those of other classes in the FIOM. Based on the results of this first stage, the integrator then marks those classes he thinks candidates for the more time intensive SEMINT process. The SEMINT tool builds discriminator vectors from the component system attributes and then provides these to the databased trained neural networks for registered classes. The degree of similarity is noted between each FIC standard representation or registered component system class and the integrator's class. This value is presented to the user as the output of the SEMINT process. The system integrator validates the results of automated correlation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

A. EFFECTIVENESS

Unfortunately, there is no empirical data to unequivocally state that the F-IDE correlation algorithm is better than any other proposed method. Since the OOMI, FIOM, and FICs are all still under investigation by Young and others, there is no baseline to measure improvements to the correlation problem. To test the effectiveness of the correlation algorithm, all component system schemas and the FIOM had to be hand-generated - a time-intensive process. As the size of the FIOM (the algorithm's search space) and the number of component system XML schemas available grow, and as other correlation methodologies are proposed, measures of effectiveness will be available. For comparison, this is the same situation that faced Clifton when he used his SEMINT tool to correlate fields in heterogeneous database. Previously the only way this matching was done was by hand. Using the SEMINT tool, Clifton was able to reduce the amount of effort from four hours per data element to twenty minutes. If this is any indication of the possible success of the F-IDE correlation tool, further research is warranted.

One obvious advantage of the proposed method of correlation is that it requires very little user intervention. We have assumed that component system schemas only contain enough information to sufficiently validate an XML message plus some metadata readily available in the data dictionaries. The task of searching the FIOM is thus simplified for the integrator since he only has to supply his component system schema. Pressing a button, deciding on a syntax match threshold, and pressing another button present him with ranked lists of possible matching classes. Nebulous browsing through

hierarchical structures and formulating convoluted Boolean searches would be more complicated and uncertain. These search methods are not dismissed, however, and the integrator constantly has the ability to drill down into the registered classes at any point in the process. This allows integrators to use their domain knowledge to "tune" the search between stages.

Currently the extent of syntax information available to the F-IDE depends greatly on the developer of the component system schemas. We have assumed that this information will be limited to what is readily available in the schema. Obviously the more metadata system designers include about their entity, the better the chance for a successful match. As these detailed schemas are incorporated into the FIOM, this will improve the overall precision of the search algorithm.

Despite the fact that SEMINT claims to reveal "semantics" of an attribute, it actually gathers syntactical information, which supposedly reveals the attribute's semantics. In order to make a successful match, then, **both attributes must have a similar syntax or representation in their schema**. This is contrary to our belief that entities will have heterogeneity of representation between systems. This means that the SEMINT discriminator signatures for two entities with identical semantics will be different if they are represented differently in any way. Although not supported with empirical data, it is assumed that within the limited domain of military messages, there is enough commonality between systems that heterogeneity of representation will be an anomaly and relatively insignificant in determining final search results.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

The F-IDE SEMINT tool currently draws all its information from the component system's XML schema. This technique falls short of the full use of SEMINT because it ignores data contents. In the original tool, once the database fields were determined, the algorithm used statistical information about the actual contents of those fields as additional discriminators. If statistical information could be gathered on the actual XML documents being produced and consumed by the system, it would give a more detailed picture of the semantics of the entities involved. For example, while a "PRODUCED_BY" attribute may have a maximum field length of twenty-five characters, in use its average length may only be ten. On the other hand, a "TIME" field may have a maximum length of four numeric characters and have and average length of four characters. This is only one example. Clifton lists twenty characteristics for use in database integration efforts. While not all of these are applicable for general entity correlation, they provide a good starting point for future research in using SEMINT with the FIOM.

Another area where recent research into improving search algorithms might aid the F-IDE correlator is in syntactic searches. The current prototype of the system uses a very naïve algorithm to decide on a keyword score. There are better ways already developed for executing this kind of search including using a thesaurus and calculating degrees of similarity between words, or using relative position of words within the document. Any improvement in the precision of the syntax-checking step of the correlator will translate to a better score for the SEMINT step. If system designers

improve their XML schemas and make them more text-intensive, an improved syntax correlation step might have significant impacts on the overall methodology.

C. CONCLUSIONS

The correlation of entities between heterogeneous systems is currently done by hand and, therefore, the success of an automated tool to assist this effort is difficult to measure. The SEMINT tool promises to aid system integrators by automatically determining a wide range of statistical information about each attribute. By examining XML schemas for message formats, SEMINT understands what kind of data the system designers anticipated for each field. When implemented, a "message listener" for intercepting actual XML messages being passed from producer to consumer, will allow SEMINT to generate statistics for the data that is *actually* passed in these fields. This will greatly enhance the precision of the correlation algorithm.

We predict XML messages will be the choice for DoD messaging in the near future. This will lead to improvements to interoperability and enhanced information sharing. The F-IDE will allow system integrators to use the XML schemas and XML messages, readily available, to build bridge languages between heterogeneous systems. Interoperability will result in a drastic improvement in the speed, quality, and quantity of information available to the American warfighter. The end result will be fewer friendly casualties and an overall better chance for success in future conflicts.

APPENDIX A: DETAILED XML SCHEMA (PARTIAL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 by Paul Young (NPS) -->
<xsd:schema targetNamespace="http://www.nps.navy.mil/namespace/usmtf"
xmlns:usmtf="http://www.nps.navy.mil/namespace/usmtf"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="qualified">
  <xsd:element
name="artillery_target_intelligence_target_information_request">
    <xsd:annotation>
      <xsd:documentation>The ATI.ATR is used to exchange artillery
target information among the fire support facilities of a joint task
force (JTF) or combined force.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="exercise_identification">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="exercise_nickname">
                  <xsd:simpleType name="exercise_nickname_type">
                    <xsd:annotation>
                      <xsd:documentation>The unique code name or
nickname assigned to a joint/combined exercise or plan or to designate
exercise message traffic.
                      </xsd:documentation>
                    </xsd:annotation>
                    <xsd:restriction base="xsd:string">
                      <xsd:minLength value="1"/>
                      <xsd:maxLength value="56"/>
                    </xsd:restriction>
                  </xsd:simpleType>
                </xsd:element>
              </xsd:sequence>
              <xsd:attribute name="setid" type="xsd:string" use="fixed"
value="EXER"/>
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="operation_identification_data">
            <xsd:complexType
name="operation_identification_data_settype">
              <xsd:sequence>
                <xsd:element name="operation_codeword"
type="usmtf:operation_codeword_fieldtype"/>
              </xsd:sequence>
              <xsd:attribute name="setid" type="xsd:string" use="fixed"
value="OPER"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
        <xsd:sequence>
```

(This section omitted for clarity)

```
            <xsd:element name="day_time_group_data">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="military_day_time">
                    <xsd:complexType>
                      <xsd:annotation>
                        <xsd:documentation>The day of a month and
timekeeping in hours and minutes of a calendar day, using the 24-hour
clock system and an associated time zone.
                        </xsd:documentation>
                      </xsd:annotation>
                      <xsd:sequence>
                        <xsd:element name="day">
                          <xsd:simpleType name="day_type">
                            <xsd:restriction base="xsd:string">
                              <xsd:length value="2"/>
                              <xsd:pattern value="\[0-3]{1}[0-9]{1}"/>
                            </xsd:restriction>
                          </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="hour_time">
                          <xsd:simpleType name="hour_time_type">
                            <xsd:restriction base="xsd:string">
                              <xsd:length value="2"/>
                              <xsd:pattern value="\[0-2]{1}[0-9]{1}"/>
                            </xsd:restriction>
                          </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="minute_time"
type="usmtf:minute_time_type"/>
                        <xsd:element name="time_zone"
type="usmtf:time_zone_type"/>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="setid" type="xsd:string" use="fixed"
value="DTG"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        <xsd:element name="decloadr"
type="usmtf:message_downgrading_or_declassification_data_settype"
minOccurs="0"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

APPENDIX B: SAMPLE KEYWORD LIST

<?xml version="1.0" encoding="UTF-8"?>
<keywords>artillery target intelligence artillery target report MSGID OPT TNO DTG GRID GZE TST message type originator primary option target number date time group UTM easting UTM northing altitude grid zone target type target subtype degree of protection target number prefix target number suffix day hour minute message identification asca option target number date time target location asca grid zone target description message type asca originator primary option target number fieldtype asca date time group UTM easting UTM altitude grid zone asca target type asca target subtype degree of protection asca target number prefix type asca target number suffix type asca day type asca hour time type asca minute time type The ATI.ATR is used to transmit target information in the form of complete target records either on the initiative of the sender or in response to ATI.TIR one-time requests for information. Message Identification Option Target Number Date-TIme Target Location Grid Zone Target Description Seven characters used to specify the message type. Thirteen characters used to specify the logical name of the message originator. Three (3) letters used to specify the primary option to be taken. Legal entries CAN- Cancel; XMT- Transmit. Six (6) characters used to specify the target number. The first two (2) characters shall be letters followed by four (4) digits. Six (6) digits used to specify the ZULU date-time that the target was acquired or the date-time that the target data was updated. The first two (2) digits represent the day of the month, the second two (2) digits the hour of the day, and the final two (2) digits, the minutes of the hour. Six (6) digits used to specify the higher order Easting of the target location in meters. One (1) to eight (8) digits used to specify the higher order Northing of the target location in meters. One (1) to five (5) characters used to specify the altitude of the location in meters. Altitudes below sea level shall be preceded by a minus (-) sign. Plus (+) is understood if a minus (-) is not specified. One (1) to three (3) characters used to specify the earth hemisphere and grid zone designator. One (1) or two (2) digits used to specify the grid zone at the location proceeded by one (1) character used to specify the earth hemisphere. Grid zones in the southern hemisphere shall be preceded by a minus (-) sign. Plus (+) is understood if a minus (-) is not specified but may be entered. Three (3) to six (6) letters used to specify the target type. See Legal Entries Table 11 (Legal Entries for Target Type/Subtype), page 405 for the codes. Two (2) to six (6) letters used to specify the target subtype. See Legal Entries Table 11 (Legal Entries for Target Type/Subtype), page 405 for the codes. Four (4) to six (6) letters used to specify the degree of personnel protection. See Legal Entries Table 12 (Legal Entries for Degree of Protection), page 407 for the codes. Comment describing your root element message type originator primary option UTM easting UTM northing altitude grid zone target type target subtype degree of protection target number prefix type target number suffix type day type hour time type minute time type message identification settype option settype target number settype date time settype target location settype grid zone settype target description settype target number date time group CAN XMT</keywords>

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: JAVA CODE - KEYWORD GENERATOR

```
/**
 * Title:       KeywordMaker
 * Description: This class allows users to pass a DOM document and parse through
 *              it for the identifying terms (documentation elements, element
 *              names, etc). The user can access this list of terms by using
 *              the provided api.
 * Copyright:   Copyright (c) 2001
 * Company:     US Naval Postgraduate School
 * @author Randy Pugh
 * @version 1.0
 */

import org.w3c.dom.*;

public class KeywordMaker
{
 // classwide variables
 protected Document doc;
 protected StringBuffer wordList;


/** Title:          KeywordMaker()
 *  Description:    Default Constructor
 *  Parameters:     None
 *  Return Value:   None
 */
 public KeywordMaker()
 {
 }


/** Title:          KeywordMaker(Document d)
 *  Description:    Constructor which sets the underlying dom and gets the
 *                  relevant keywords from the model.
 *  Parameters:     Document d => w3c dom representing schema to get keywords
 *                  from.
 *  Return Value:   None
 */
 public KeywordMaker(Document d)
 {
  doc = d;
  wordList = new StringBuffer();
  makeListOfTerms();
```

```
 } // end of KeywordMaker


/** Title:          setDocument(Document d)
 * Description:    Sets underlying model and gets the list of keywords.
 * Parameters:     Document d => underlying w3c dom for gathering keywords
 * Return Value:   None
 */
 public void setDocument(Document d)
 {
  doc = d;
  wordList = new StringBuffer();
  makeListOfTerms();
 }

/** Title:          makeListOfTerms()
 * Description:    Adds the user's preferred terms to the stringbuffer. Calls
 *                 on addTerms to append either data or attribute values. this
 *                 section can easily be modified to expand or restrict the
 *                 terms added to the keyword list
 * Parameters:     None
 * Return value:   None
 */
 private void makeListOfTerms()
 {
  Node root = doc.getDocumentElement();
  // get all the element names
  addTerms("xsd:element", "name");
  addTerms("xsd:element", "type");
  addTerms("xsd:documentation");
  addTerms("xsd:attribute", "name");
  addTerms("xsd:enumeration", "value");
 } // end of makeListOfTerms


/** Title:          addTerms(String tag)
 * Description:    Walks through the DOM and gets the tags requested and then
 *                 adds the data between the tags to the list of keywords
 * Parameters:     String tag => tag whose value represents a keyword
 * Return value:   None
 */
 private void addTerms(String tag)
 {
  NodeList tagList = doc.getElementsByTagName(tag);
  // go through every tag...
```

```
    for (int i = 0; i < tagList.getLength(); i++)
    {
     // get the children so we can get the text node
     NodeList childList = tagList.item(i).getChildNodes();
     for (int j = 0; j < childList.getLength(); j++) {
      if (childList.item(j).getNodeType() == Node.TEXT_NODE)
      {
       String nodeValue = childList.item(j).getNodeValue();
       if (nodeValue != null)
       {
        wordList.append(nodeValue + " ");
       }
      } // end if text node
     } // end loop through children of tag
    } // end loop through tags
  } // end of addTerms procedure


/** Title:        addTerms(String tag, String attrName)
 * Description:   Walks through the DOM and gets the tags requested and then
 *                finds the attribute requested and adds the value of that
 *                attribute to the list of keywords
 * Parameters:    String tag => tag who has an attribute representing a keyword
 *                String attrName => attribute name whose value is a keyword
 * Return value:  None
 */
 private void addTerms(String tag, String attr)
 {
   NodeList tagList = doc.getElementsByTagName(tag);
   // go through every tag...
   for (int i = 0; i < tagList.getLength(); i++)
   {
    // get the children so we can get the text node
    NamedNodeMap attrList = tagList.item(i).getAttributes();
    for (int j = 0; j < attrList.getLength(); j++) {
     if (attrList.item(j).getNodeName().compareTo(attr) == 0)
     {
      String nodeValue = attrList.item(j).getNodeValue();
      if (nodeValue != null) wordList.append(nodeValue + " ");
     } // end if text node
    } // end loop through children of tag
   } // end loop through tags
 } // end addTerms procedure
```

```
/** Title:        getKeywords()
 * Description:   Returns the user a string consisting of the representative
 *                words and terms extracted from the schema passed during
 *                instantiation. Removes underbars ('_') and colons (':') to
 *                allow for variable naming conventions joining keywords
 * Parameters:    None
 * Return value:  String => keywords contained in the underlying w3c document
 */
public String getKeywords()
{
  String words = wordList.toString();
  words = words.replace('_', ' '); // remove underbars
  words = words.replace(':', ' '); // remove :'s (used in namespace references)
  return words;
} // end getKeywords procedure

} // end of KeywordMaker class definition
```

LIST OF REFERENCES

[DKM+93] Drew, P., King, R., McLeod, D., Rusinkiewicz M., Silberschatz, A., "Report of the workshop on semantic heterogeneity and interoperation in multi-database systems," *SIGMOD Record*, pp.47-56, September 1993.

[LC94] Li, W., Clifton, C, "Semantic Integration in Heterogeneous Databases Using Neural Networks," *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB-94)*, pp.1-12, 1994.

[Clif99] Clifton, C., Housman, E., Rosenthal, A., "Experience with a Combined Approach to Attribute-Matching Across Heterogeneous Databases," *IFIP DS-7 Data Semantics Conference*, 1997.

[GN86] The Goldwater-Nichols Department of Defense Reorganization Act of 1986 (Public Law 99-433).

[Koh87] Kohonen, T., "Adaptive, Associative, and Self-Organizing Functions in Neural Computing," *Applied Optics*, 26:4910-4918, 1987.

[Liskov87] Liskov, B., "Data Abstraction and Hierarchy," *OOPSLA 1987, Addendum to Proceedings*, 1987.

[Meye88] Meyer, B., *Object-Oriented Software Construction*, Prentice Hall, 1988.

[Prie85] Prieto-Diaz, R., "Implementing Faced Classification for Software Reuse," *Communications of the ACM*, v. 34, pp.88-97, May 1991.

[SK92] Sheth, A., Kashyap, V., "So Far (Schematically) yet So Near (Semantically)," *Interoperable Database Systems (DS-5)*, 1993.

[SM83] Salton, G., McGill, M., *Introduction to Modern Information Retrieval.* McGraw Hill, New York, 1983.

[Stei91] Steigerwald, R., "Reusable Software Component Retrieval Via Normalized Algebraic Specifications," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, 1991.

[Web97] Webster's Basic English Dictionary

[Wie93] Wiederhold, G., (editor), *Intelligent Integration of Information*, Kluwer Academic Publishers, Boston MA, July 1996.

[Wood85] Wood, J., "What's in a link?," *Readings in Knowledge Representation*, Morgan Kaufmann, 1985.

[Young01] Young, P., "Achieving Interoperability of Heterogeneous Software Systems Through Computer-Aided Resolution of Data Representation Differences," Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, Unpublished.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Director, Training and Education 1
   MCCDC, Code C46
   1019 Elliot Road
   Quantico, VA 22134-5027

4. Director, Marine Corps Research Center 2
   MCCDC, Code C40RC
   2040 Broadway Street
   Quantico, VA 22134-5107

5. Director, Studies and Analysis Division 1
   MCCDC, Code C45
   300 Russel Road
   Quantico, VA 22134-5130

6. Chairman 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA 93943

7. Professor D. C. Boger, Code CC 1
   Naval Postgraduate School
   Monterey, CA 93943

8. Professor V. Berzins 1
   Department of Software Engineering
   Naval Postgraduate School
   Monterey, CA 93943

9. CAPT Paul E. Young 1
   Department of Software Engineering
   Naval Postgraduate School
   Monterey, CA 93943

10. Randy Pugh 1
216 Edgewater Drive
Edgewater, MD 21037